

Les Nouvelles d'APL

Revue Trimestrielle d'AFAPL

(Association Francophone pour la promotion du langage APL)

N°30 Juillet 1999

***Dans ce numéro* :**



Les Nouvelles d'APL

Revue Trimestrielle d'AFAPL

(Association Francophone pour la promotion du langage APL)

N°30 Juillet 1999

Dans ce numéro **:**

AFAPL

Association Francophone pour la promotion du langage APL
Association régie par la loi de 1901
174 bd. de Charonne - 75020 PARIS - FRANCE
Tél. & Fax : (33) 01-43-56-31-79
URL = http://www.ensmp.fr/~scherer/langlet/

Constitution du bureau

Président	M. Bernard Legrand
Trésorier	M. Bernard Secret
Trésorier adjoint	M. Raymond Tisserand
Secrétaire G[al]	Mme Ludmila Lemagnen
Secrétaire adj.	M. Bernard Mailhol

Autres membres du Bureau :

M. Eric Lescasse
M. Sylvain Baron
M. Gérard Bousquet
M. Raymond Coquidé

Revue "Les Nouvelles d'APL"

Imprimée par AFAPL
Dépôt légal : juin 1996
Numéro ISSN : 1664-4699
Directeur de la Publication :
M. Bernard Legrand
Rédaction Générale :
Michel J. Dumontier

Comment devenir Membre d'AFAPL

Faire parvenir votre Bulletin d'Adhésion (voir à la fin de ce numéro) rempli, accompagné d'un chèque bancaire couvrant la cotisation annuelle au Secrétariat ou directement au Trésorier :

Mme Lemagnen
Secrétaire G[al] d'AFAPL
174 bd. de Charonne
75020 PARIS

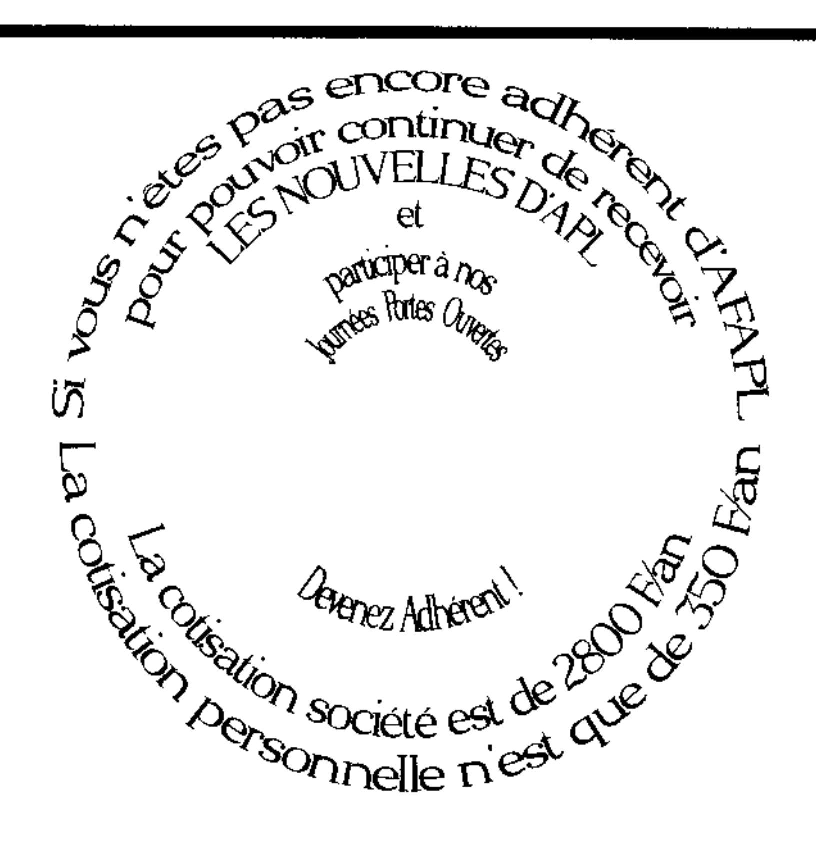
M. B. Secret
Trésorier d'AFAPL
35 r. Jules Ferry
91200 ATHIS-MONS

Cotisation	Montant
Personnelle	350 F
Société	2800 F

Ajouter 100F pour un abonnement à l'étranger. La cotisation **Société** donne droit à figurer gratuitement dans le Catalogue des Produits et Services APL

Tarif des publicités (*voir* : **Calendrier des prochaines parutions**)

Sommaire



La cotisation personnelle donne droit à:

- recevoir un exemplaire des **Nouvelles d'APL** (par trimestre)
- participer aux 1/2 journées Portes Ouvertes
- adresser un CV pour publication, en cas de recherche d'emploi

La cotisation Société donne droit à:

- recevoir 5 exemplaires des **Nouvelles d'APL** (par trimestre)
- être listé dans notre "Guide des Produits et Services APL"
- pouvoir inclure des Publicités (1000 F par page, 600 F par 1/2 page)
- participer aux 1/2 journées Portes Ouvertes

Editorial du Président

par Bernard Legrand

Chaque nouveau membre de l'Académie Française se doit de faire l'éloge de celui auquel il succède. Nous ne sommes pas, hélas, l'Académie, mais venant de succéder à Bernard Mailhol à la présidence de notre association, je mesure l'ampleur de la tâche.
Car enfin, on peut se demander :
"Pourquoi une association pour la promotion d'APL ?"
Il n'y a pas, que je sache, d'association pour la promotion de COBOL, pas plus que de C++ ni de JAVA. Pourquoi APL ferait-il exception ?
Les premiers congrès APL se tenaient dans la liesse, avec l'espoir de voir ce séduisant langage conquérir des esprits supposés cartésiens et enthousiastes... Le temps a passé, et la stratégie conquérante s'est muée peu à peu en tactique défensive, les associations d'APListes successives défendant un dernier carré chaque fois plus réduit.
Serions-nous en train de vivre une sorte de "Fort Alamo" ou de "Camerone", et de livrer bataille à des hordes d'envahisseurs sauvages et incultes ? Même pas !
Fini le temps des agressions contre APL, dans lesquelles nous puisions la force de faire toujours mieux : le dernier carré de fidèles vieillit et s'étiole dans l'indifférence générale : on nous ignore.
On n'attaque plus APL parce qu'il ne dérange plus : APL serait-il en état de mort clinique ?
Ma réponse est multiple :

1 – Une notation algébrique ne saurait ni vieillir ni mourir, et nous devrions être confiants dans sa pérénnité.
2 – Mais les produits les mieux conçus, les plus intelligents, meurent vite s'ils ne sont pas relayés par un succès commercial marquant. Ils sont balayés par des produits parfois médiocres mais puissamment supportés. N'est-ce pas Bill ?
3 – Qu'ai-je fait,
qu'as-tu fait,
qu'a-t-il (ou elle) fait,
qu'avons-nous fait pour faire vivre APL ? Chacun de nous doit se poser la question.

Les quelques membres qui, comme moi, vivent de l'APL, croient avoir fait de leur mieux, et ce n'est pas par hasard que c'est en leur sein qu'on a puisé les derniers présidents d'AFAPL. Mais avons-nous vraiment TOUT fait ce qui était à portée de nos modestes moyens ?
Quoi qu'il en soit, ce n'est pas suffisant. Pour que l'association vive, il faut que ses membres puissent s'y reconnaître, et s'en nourrir. Pourquoi paieriez-vous 350 Francs si vous n'obtenez rien de concret en échange ?

C'est la raison pour laquelle ma première décision a été d'adresser à tous les membres actuels ou passés une enquête sur ce que vous attendez de l'association. A l'heure où j'écris cet éditorial, nous avons reçu ... deux réponses ! Comment faire évoluer l'association sans le concours actif de ses membres ?
Si vous ne l'avez déjà fait, renvoyez cette enquête, ou faites nous part de suggestions plus personnelles. Faites-nous connaître l'avis de personnes qui lisent la revue sans être membres eux mêmes. Nous en avons besoin. Je m'engage à vous en rendre compte dans un prochain numéro
Avec vous, j'aimerais revitaliser APL.
Et vous, qu'aimeriez-vous faire ?
Passez un agréable été.

Bernard Legrand
21 Juin 1999

Compte d'activité de l'année 1998

Par Bernard Secret

	1993	1994	1995
RECETTES			
Cotisations entreprises	29 300,00	33 600,00	28 000,00
Cotisation individuelles	32 400,00	34 575,00	22 925,00
Contributions volontaires	400,00	550,00	175,00
Publicité	9 769,53	2 599,13	2 412,00
Total des recettes	**71 869,53**	**71 324,13**	**53 512,00**
DEPENSES			
Frais administratifs	7 915,63	1 135,10	439,00
Télécommunications	3 816,83	3 604,53	2 306,58
Publications	42 793,48	54 844,92	49 371,08
Divers	1 093,00	0,00	0,00
Frais financiers	163,95	163,95	0,00
Matériels	0,00	3 890,00	4 580,00
Total des dépenses	**55 782,89**	**63 638,50**	**56 696,66**
SOLDE DE GESTION	**16 086,64**	**7 685,63**	**-3 184,66**

	1996	1997	1998
RECETTES			
Cotisations entreprises	22 400,00	16 800,00	27 440,00
Cotisation individuelles	18 848,00	15 750,00	16 050,00
Contributions volontaires	0,00	0,00	0,00
Publicité	0,00	0,00	0,00
Total des recettes	**41 248,00**	**32 550,00**	**43 490,00**
DEPENSES			
Frais administratifs	263,90	916,71	225,50
Télécommunications	3 935,56	3 899,87	4 761,26
Publications	30 343,91	28 129,45	11 553,29
Divers	1 000,00	792,54	2 252,68
Frais financiers	0,00	0,00	
Matériels	0,00	3 790,00	
Total des dépenses	**35 543,37**	**37 528,57**	**18 792,73**
SOLDE DE GESTION	**5 704,63**	**-4 978,57**	**24 697,27**

Génération de fonctions polynomiales ayant la propriété F(x)=F(1/x)
1ère partie : démonstration et fonctions APL.

Par Michel J. DUMONTIER

Introduction : Le journal est intitulé les nouvelles d'APL, et pourtant je vais faire resurgir le passé (1960 : presque 40 ans !, APL n'était pas encore né !) en actualisant par APL, donc en APLisant, des propriétés que j'avais découvertes en rédigeant un D.S. de Taupe (sujet de concours d'entrée aux 'grandes écoles') que le digne professeur sorti major d'Ulm n'avait pas compris mais qui a fini par comprendre lorsque, sur sa demande, je lui ai expliqué après le cours de maths !
Cette propriété, qui n'avait pas été demandée d'être démontrée, m'avait permis de faire des gros raccourcis dans les démonstrations demandées…ce qui m'avait permis, comme d'habitude, de finir et sortir avant les 4 heures fatidiques !
Anecdote 1: J'ai conservé la démonstration, mais je n'ai pas retrouvé le D.S. que j'avais mis dans un cartable vert format A1 avec tous mes exploits et que j'ai laissé dans un hôtel à Malakoff dans les années 70. Si quelqu'un le retrouve…ou l'a emprunté comme on dit maintenant avec l'évolution des mœurs, on ne sait jamais !
Anecdote 2 : j'avais promis à Gérard Langlet d'écrire ce présent article et de lui envoyer , mais j'ai toujours différé la rédaction, avec l'arrière pensée que, si je le publiais, Gérard écrirait une dizaine d'articles à la suite pour parler de ce sujet comme il l'a fait pour le Quinto…
La suite, ce sera pour le prochain numéro… !

1) Démonstration :

Condition nécessaire et suffisante pour qu'un polynôme P(x) soit tel que
$P(x) \equiv x^r P(1/x)$

(Je reproduis le texte strictement comme je l'ai écrit il y a 40 ans !)

Soit un polynôme $P(x)=a_n x^n+a_{n-h}x^{n-h}+\ldots+a_{q+h'}x^{q+h'}+a_q x^q$ ordonné suivant les puissances décroissantes de x et où on symbolise les termes de rang h à partir du début et de rang h' à partir de la fin par $a_{n-h}x^{n-h}$ et $a_{q+h'}x^{q+h'}$ et où le terme de plus bas degré a pour degré q.
Formons $P'(x)= x^r P(1/x)= a_n x^{r-n}+a_{n-h}x^{r-n+h}+\ldots+a_{q+h'}x^{r-q-h'}+a_q x^{r-q}$

Rangeons ce polynôme suivant les puissances décroissantes :

$x^r P(1/x)= a_q x^{r-q} + a_{q+h'} x^{r-q-h'} + \ldots + a_{n-h} x^{r-n+h} + a_n x^{r-n}$

Condition Nécessaire :

identifions les termes de plus haut degré :

$a_q = a_n \qquad r-q=n \quad \rightarrow \quad r=n+q$ \

identifions ceux de plus bas degré : > c'est la même condition.

$a_n = a_q \qquad r-n=q \quad \rightarrow \quad r=n+q$ /

Rang des termes pour que les degrés soient les mêmes :

$n-h=r-q-h' \rightarrow n-h=n+q-q-h' \rightarrow -h=-h' \rightarrow h=h'$

d'où : $a_{n-h} = a_{q+h}$

Le polynôme devra donc être de la forme :

$P(x)=a_n x^n + a_{n-h1} x^{n-h1} + \ldots + a_{n-h1} x^{q+h1} + a_n x^q$

Où les coefficients équidistants des extrêmes sont égaux et où, à chaque terme de coefficient a_{n-hi} et de degré n-hi correspond un terme et un seul de même coefficient a_{n-hi} et de degré q+hi avec $0<hi\leq p$ et où p est tel que si le nombre de termes est pair et égal à 2N → $p\leq N-1$ et si le nombre de termes est impair et égal à 2N+1 → $p\leq N$ dans ce dernier cas, il n'y a qu'un terme pour lequel h=N si celui-ci existe.

Condition suffisante :

Soit un polynôme de la forme :

$P(x)=a_n x^n + a_{n-h1} x^{n-h1} + a_{n-h2} x^{n-h2} + \ldots + a_{n-h2} x^{q+h2} + a_{n-h1} x^{q+h1} + a_n x^q$

$x^{n+q} P(1/x)= a_n x^q + a_{n-h1} x^{q+h1} + a_{n-h2} x^{q+h2} + \ldots + a_{n-h2} x^{n-h2} + a_{n-h1} x^{n-h1} + a_n x^n$

donc :

$x^{n+q} P(1/x) \equiv P(x)$

Et c'est ici que cela devient interessant :

PROPRIÉTÉS IMMÉDIATES :

Soit une fraction g(x) qui est le quotient de 2 tels polynômes tels que

$\varphi(x)=x^{n+q}\,\varphi(1/x)$ et
$f(x)=x^{m+\lambda}\,f(1/x)$ on a

$g(x)=\varphi(x)/f(x) = x^{n+q}/x^{m+\lambda}\,g(1/x)$

En particulier si $n+q=m+\lambda$ alors

$$\boxed{g(x)\equiv g(1/x)}$$

C.Q.F.D.

2) Implantation en APL.

Comme je le disais plus haut, n'ayant plus la famille de polynômes étudiés en prépa, il va falloir inventer des polynômes adéquats pour commencer.

Voici déjà quelques fonctions utiles de calcul de polynômes.

Valeur d'un polynôme de coefficients C pour la valeur X

```
Z←C P X
Z„X⊥C
```

Exemple : Valeur du polynôme $x^5+3x^4+5x^2+11$ pour x=2

```
      1 3 0 5 0 11 P 2
111
```

(J'espère que les non aplistes qui voient cela pour la 1ère fois vont apprécier, ou seront abattus, ou seront époustouflés ou ... surtout si je leur dis que ce simple symbole antitruc utilisé pour la fonction "base", sert en premier lieu à calculer un nombre en bases multiples ; je sais, beaucoup répondront : on n'a pas appris cela à l'école ! tant pis pour eux, c'est aussi l'évolution des mœurs qui veut cela !)
Fonctions de base de calculs de polynômes :

Pour les fainéants et ceux qui ont l'esprit vectoriel (et non scalaire comme les non aplistes), voici la fonction PS qui calcule les valeurs d'un polynômes pour plusieurs valeurs de x.

```
Z←C PS X ;⎕IO
⎕IO←0 ⋄ Z←(X∘.*⌽⍳⍴C)+.×C
```

Exemple :

```
1 3 0 5 0 11 PS 2  4  7
111 1883 24266
```

PLUS, MOINS, FOIS (produit de polynômes)

```
Z„X PLUS Y
Z←((-Z)↑X)+(-Z←(⍴,X)⌈⍴,Y)↑Y

Z←X MOINS Y
Z←X PLUS -Y

Z←X FOIS Y
→(0=NR Y)/3
Z←(X×¯1↑Y) PLUS (X FOIS ¯1↓Y),0 ⋄ →0
Z←0

      1 0 2 0 0 4 PLUS 3 0 2 2
1 0 5 0 2 6
```

Un gros polynôme MOINS un petit :

```
1 2 3 4 7 MOINS 2 5 4
1 2 1 ¯1 3
```

Le contraire :

```
2 4 6 MOINS 1 4 2 6 3
¯1 ¯4 0 ¯2 3
```

Rappel du binôme de Newton :

```
1 2 1 FOIS 1 3 3 1
1 5 10 10 5 1
```

Passons aux divisions : (cela nous intéresse pour le présent sujet).

```
Z←X DIVC Y;R9
→((NR X)>NR Y)/3
Z←R9, X DIVC ¯1↓Y MOINS X×R9←(¯1↑Y)÷¨1↑X ◇ →0
Z←⍳0

Z←X DIVD Y
Z←(⌽X) DIVC ⌽Y
```

(Quand on est fainéant : il faut avoir des idées !)

```
      1 3 3 1 DIVC 1 5 10 10 5 1
1 2 1
      1 3 3 1 DIVD 1 5 10 10 5 1
1 2 1
```

Fonctions annexes :

On utilise la fonction (prudente) NR dans DIVC ; la voici :

```
Z←NR X
Z←+/X=X
```

Si un polynôme calculé a une succession de coefficients nuls dans les plus hauts degrés, il faut les supprimer, c'est ce que fait la fonction SUP suivante qu'on utilisera dans les restes de divisions.

```
Z←SUP C
Z←(+/∧\0=C)↓C
```

Examples de divisions avec restes :

```
      C←1 2 1 ◇ E←9 7 10 10 5 1
      C DIVC E
1 3 3 1
      ⎕←RESTE←E MOINS C FOIS C DIVC E
8 2 0 0 0 0
      (1 2 1 FOIS 1 3 3 1) PLUS 8 2 0 0 0 0
9 7 10 10 5 1
      C DIVD E
9 ¯11 23 ¯25
      ⎕←RESTE←E MOINS C FOIS C DIVD E
0 0 0 0 32 26

      SUP RESTE
32 26
      (1 2 1 FOIS 9 ¯11 23 ¯25) PLUS 32 26
```

```
9 7 10 10 5 1
```

Une dernière fonction pour compléter : calcul des coefficients d'un polynôme dont on donne les racines.

```
Z←CPOLR R;⎕IO;A
⎕IO←0 ⋄ Z←1,⌽((⍳⍴R)∘.=+⌿~A)+.×(R)×.*A←((⍴R)⍴2)⊤⍳2*⍴R

CPOLR 2 3 5
1 ¯10 31 ¯30
```

Vérifions :

```
(1 ¯2) FOIS (1 ¯3) FOIS 1 ¯5
1 ¯10 31 ¯30
```

Maintenant, nous sommes prêts à faire un exemple pour faire des essais :

Prenons au hasard, des polynômes symétriques de même degré.

```
      ⎕←E←C,⌽C←5?10
2 6 7 1 4 4 1 7 6 2
      ⎕←C←D,⌽D←5?10
3 2 0 4 6 6 4 0 2 3
      (C PS 2, ÷2)÷ (E PS 2, ÷2)
0.6993620415 0.6993620415
```

C.Q.F.Expérimenter !

Construisons un générateur de polynômes P(x) tels que $P(x) \equiv x^r P(1/x)$

```
Z←QH POL1SURX AN;P;N;Q;H;A;D;P1
N←¯1↑AN ⋄ Q←1↑QH ⋄ P←¯1↑QH ⋄A←¯1↓AN ⋄ D←DIFF 0,H←1↓QH ⋄Z←1↑A
→((1+⍴QH)≠⍴AN)/FIN1
→((N−P)<(Q+P))/FIN2
BO:Z←Z,((¯1+(1↑D))⍴0)⋄ A←1↓A⋄ →(0=⍴A)/SUIT1⋄ Z←Z,1↑A ⋄ D←1↓D
→(0=⍴D)/SUIT1 ⋄→BO
SUIT1:P1←Z ⋄ →((N−P)=Q+P)/SYM
Z←Z,((¯1+N−(Q+2×P))⍴0)
SYM:Z←Z,⌽P1 ⋄ Z←Z,(Q⍴0) ⋄ →0
FIN1:'REVOYEZ HQ ET AN' ⋄ →0
FIN2:'LE PLUS GROS COEFF DE LA 1ERE MOITIE EST TROP GROS'
```

Ceci va permettre aux curieux et expérimentateurs de tout poil, de s'amuser avec toutes sortes de polynômes en attendant le numéro suivant de la revue AFAPL.

Si quelqu'un fait des trouvailles il peut le faire savoir à la rédaction.

Je ne m'attends pas à quinze articles là-dessus !...

Exemple nous voulons générer le polynôme symétrique qui commence par $3x^{15}+4x^{12}+7x^{10}$ et que ça se débrouille pour trouver le reste !...

alors $n=15$, $a_n=3$, $h_1=3$, $h_2=5$, $q=3$, $a_{n-h1}=4$, $a_{n-h2}=7$

```
      ⎕←C← 3 3 5 POL1SURX 3 4 7 15
3 0 0 4 0 7 0 7 0 4 0 0 3 0 0 0
   CP2
123928
      (C P ÷2)×2*(15+3)
123928
```

Ca marche !
Bon amusement !

THE 3D DCT (Discrete Cosine Transform):

A GENERALIZATION OF THE 2D DCT

Authors:
Michel J. DUMONTIER
Patrick GERLIER : Ingénieur EP.
Dr Pascal LERAY
Image Processing & Computer Graphics Expert
FRANCE TELECOM/CNET/DIH

Introduction:

The standard, classic, and well known 2D DCT is largely used in the MPEG or JPEG world, for very efficient image compression.
But the spatio-temporal 3D DCT (2 Dimensions plus time) has not yet been tested.
The present paper is dedicated to an APL implementation of the 3D DCT, defined as a generalization of the 2D DCT.

DESCRIPTION:

In this case, instead of 8*8 image blocs, we choose 8*8*8 spatio-temporal blocs.
A direct 3D DCT transforms the input bloc in the Fourier space.
A rounded vector B is obtained from this result.

An inverse 3D DCT decompression is obtained by the reverse process.
The result bloc is compared with the original bloc V13D.
The detailed implementation is shown here:

APL implementation:

The 3D DCT transform function:

```
VT←DCT3D V
⎕IO←1
N←3 1 2
VT←0.015625×MCOS+.×N⍉(MCOS+.×N⍉(MCOS+.×N⍉V))
```

The reverse 3D DCT transform function:

```
VT←IDCT3D V
⎕IO←1
N←3 1 2
VT←(⍉MCOS)+.×N⍉((⍉MCOS)+.×N⍉((⍉MCOS)+.×N⍉V))
```

Initialization function: COEFFCOS:

```
COEFFCOS
⎕IO←0
FAC←1
PI←3.141593653589
A←FAC×cos(PI÷4)
B←FAC×cos(PI÷16)
C←FAC×cos(3×PI÷16)
D←FAC×cos(5×PI÷16)
E←FAC×cos(7×PI÷16)
F←FAC×cos(PI÷8)
G←FAC×cos(3×PI÷8)
(PI÷16)∘.×⍳8
MCOS←8 8⍴0
MCOS[0;]←8 1⍴A,A,A,A,A,A,A,A
MCOS[1;]←8 1⍴B,C,D,E,(-E),(-D),(-C),(-B)
MCOS[2;]←8 1⍴F,G,(-G),(-F),(-F),(-G),G,F
MCOS[3;]←8 1⍴C,(-E),(-B),(-D),D,B,E,(-C)
MCOS[4;]←8 1⍴A,(-A),(-A),A,A,(-A),(-A),A
MCOS[5;]←8 1⍴D,(-B),E,C,(-C),(-E),B,(-D)
MCOS[6;]←8 1⍴G,(-F),F,(-G),(-G),F,(-F),G
MCOS[7;]←8 1⍴E,(-D),C,(-B),B,(-C),D,(-E)
```

GLOBAL TEST FUNCTION: TST3D

On 8*8*8 Spatio-temporal blocs:

```
Z←TST3D
⎕IO←1
COEFFCOS
A←DCT3D V13D
B←0.01×⌊A×100
Z←IDCT3D B
Z←'F6.1'⎕FMT Z[1;;]
```

The cosine function COS:

```
z←cos v
z←2○v
```

V13D is a 8*8*8 spatio-temporal bloc
V13D is loaded with a IOTA N coefficient

The result shows that the reconstructed V13D value is equivalent to the initial value.
V13D: 8 times this bloc:

```
  1  2  3  4  5  6  7  8
  9 10 11 12 13 14 15 16
 17 18 19 20 21 22 23 24
 25 26 27 28 29 30 31 32
```

```
33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56
57 58 59 60 61 62 63 64
```

The transformed matrix: B: (B is rounded from A)

```
 91.92 ¯6.45 0 ¯0.68 0 ¯0.21 0 ¯0.06
¯51.54  0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
 ¯5.39  0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
 ¯1.61  0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
 ¯0.41  0    0  0    0  0    0  0

 ¯0.01  0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0

  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0

  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0

  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0


  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
  0     0    0  0    0  0    0  0
```

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Conclusion:

Only 9 coefficients are different from 0.
All the energy is concentrated on the first coefficients.
(As for the classic 2D DCT transform)
But in the 3D case, the compression ratio is much more efficient:

Compression ratio possible evaluation:
2 coefficients using: 16bits
1 coefficients using: 10bits
5 coefficients using 8 bits

Total: 34 bits (< 9 octets, for 8*8*8 = 512 coefficients of 16 bits)
For this example, the compression ratio is:
1024/9 = 113

FURTHER RESEARCHES:

Several other examples could be tested, mainly for real images:
One could transform an animated spatio-temporal sequence, and test with the 3D DCT transform.
Test sequences could be choosen among images with low or high frequencies.
(classic DCT beeing more efficient for low frequencies)
But adaptive quantizing could be used, with, for ewample, neural nets in order to optimized the quantizing process.

GENERAL CONCLUSION:

This first result seem very promising.

Further implementations and links with real image sequences must be done and tested now.

But at present, APL implementation demonstrates a very efficient coding compacity, able to solve very complex algorithms in a very short time, without the complex requirements of C or C++.

OPTIMISATION ET ORDINATEUR

Une tentative d'optimisation de l'optimisation
vers l'optimiseur relatif absolu,
la Méthode des Moindres Quelconques (MMQ)

Gérard A. Langlet
CEA/DSM/DRECAM/SCM/LIT (LABORATOIRE D'INFORMATIQUE THEORIQUE)

Proposition d'article, compréhensible par tout lecteur sensé et de bonne foi, pour les brèves du DRECAM, par exemple.

Introduction

La méthode des moindres carrés reste, et de loin, la plus connue et, probablement la plus utilisée, de toutes les méthodes d'optimisation. Simple à exposer, elle a conquis tous les domaines: on la retrouve utilisée par les politologues aussi bien que par les physiciens, par les analystes financiers aussi bien que les biologistes et des médecins.

Son principe initial ne fait pas appel à des mathématiques de haut niveau : il peut s'enseigner dans des classes élémentaires, dès que le représentation cartésienne des fonctions a été assimilée.

Notre but n'est pas d'exposer cette méthode de la manière classique, mais plutôt d'une façon telle qu'elle va se retrouver elle-même optimisée, si l'utilisateur désire la pratiquer sur une grande quantité de données, à l'aide d'un ordinateur et non plus à la main. Elle deviendra alors, sans effort, la plus efficace de toutes les méthodes connues, car de nouvelles propriétés vont apparaître, en raison du mode de fonctionnement de l'ordinateur, au niveau où il gère de façon optimale, donc sans aucune erreur, l'information qu'on lui soumet, celui de l'algèbre binaire.

La Base

Soit une fonction Y(t)=F(Xl, X2, ... ,Xn, t) exprimant, en fonction de différents paramètres ajustables Xi ... Xn, l'évolution d'un phénomène quelconque en fonction de t, le temps.

En réalité, les différentes valeurs de Y peuvent se représenter sur un graphe cartésien en fonction de la variation de n'importe quel paramètre au choix, t ne représentant alors qu'un choix particulier (très courant toutefois, aussi bien en physique qu'en biologie). L'examen de Y=F(t) suffit alors pour exposer la base

de la méthode, sur ce que l'on appelle une série chronologique.

Supposons que F exprime l'état actuel d'une théorie, c'est-à-dire la meilleure fonction connue capable d'exprimer, le plus correctement possible, et pour des intervalles raisonamment larges de variation de tous les autres paramètres X1 ... Xn, la variation de F en fonction de t.

Pour des valeurs connues de tous les paramètres, l'utilisateur va alors tracer, sur une feuille de papier ou un écran graphique, le graphe de Y=F(t), après avoir calculé un certain nombre, toujours fini, de valeurs de Y, pour le même nombre de valeurs de t. Appelons Yt cette série de nombres.

Parallèlement, l'utilisateur a effectué des mesures avec le plus grand soin, pour ces mêmes valeurs de t, en fixant les valeurs des autres paramètres aux mêmes valeurs que celles utilisées pour les calculs. Appelons Ym cette série de nombres.

Remarque. Il n'est pas nécessaire, pour la suite de l'exposé, que l'on ait fixé des valeurs constantes pour l'ensemble des autres paramètres lorsque Y varie en fonction de t; il est seulement indispensable que, pour chaque calcul, les paramètres intervenant dans F aient tous la même valeur que ceux utilisés au même point pour effectuer la mesure. Yt et Ym représentent alors deux trajectoires, l'une théorique, l'autre expérimentale, car mesurée, dans l'hyperespace Y=F(Xl, ... Xn) en fonction de t. Le raisonnement ci-après va alors acquérir un caractère très général.

Moindres carrés et autres moindres.

On peut représenter sur le même papier ou écran les deux graphes de Yt et Ym en fonction de t en utilisant des coordonnées cartésiennes, d'une manière très classique; c'est d'ailleurs ce que font la plupart des utilisateurs et les auteurs de publications ou d'ouvrages didactiques, quel que soit le domaine d'application considéré : un graphique bien présenté parle toujours plus que de longs et austères tableaux de nombres. L'information se transmet alors de manière visuelle.

En général, aucune théorie n'est complète... Il sera extrêmement rare que tous les points d'ordonnées Ym coïncident parfaitement avec les points d'ordonnées Yt pour chaque valeur de t. Même si la coïncidence dans le cas étudié est suffisamment bonne partout pour valider la théorie, il va apparaître des écarts si on essaie a) de prévoir le devenir du système étudié (tel est le but de toutes les séries chronologiques), au delà des valeurs de t mesurables et effectivement mesurées, b) de changer de trajectoire en modifiant les valeurs des autres

paramètres du système, c) d'affiner les mesures en contrôlant mieux la précision de tous les paramètres.

Chaque point des graphes de Ym et de Yt doit, honnêtement, devenir le barycentre (centre de gravité) d'une fenêtre rectangulaire, dont la longueur et la largeur de part et d'autre du point central, exprimeront les erreurs absolues sur la détermination théorique et expérimentale de t et des valeurs de Y.

Dans le cas général, les erreurs sur t sont négligeables devant celles sur Y. Avec la précision dont on dispose pour le calcul théorique de Yt sur ordinateur, (mais pas toujours si F est une fonction très complexe exigeant des millions d'itérations par exemple), les erreurs sur Yt sont négligeables devant celles sur Ym, sauf si la formule est vraiment fausse...

Considérons seulement le cas où le rectangle sur chaque point de Yt se réduit effectivement à un point, donc, sur l'écran de visualisation, à un pixel élémentaire (de l'anglais "picture element", petit carré noir rempli par le point visible sur l'écran, et qui n'est jamais un point euclidien (théoriquement sans dimension). Le rectangle construit sur chaque point de Ym, puisque l'épaisseur (largeur) sur t vaut aussi 1 pixel, devient visuellement représentable par des flèches verticales opposées

↑ l'espace blanc entre les flèches ayant alors aussi
↓ 1 pixel de largeur et 1 pixel d'épaisseur.

Si théorie et mesures coïncident exactement, c'est-à-dire au mieux pour la précision affichable sur l'écran, le pixel théorique (noir) vient boucher le trou entre les deux flèches, exactement. Si l'accord est acceptable, le pixel noir théorique devient invisible, car il se situe dans l'une des flèches, soit en haut, soit en bas. Il suffit alors, sur un écran polychrome, de lui affecter par exemple une couleur verte pour le voir néanmoins; sur un écran monochrome, on peut, soit remplacer le pixel noir théorique par un marqueur (croix, carré, losange, petit rond, etc...), soit, encore plus simplement, inverser sa couleur de sorte que l'une des deux flèches noires apparaisse affectée d'un trou blanc.

Remarque. Les flèches n'ont pas nécessairement toutes la même hauteur pour tous les points, car la précision expérimentale peut varier d'un point à l'autre.

Lorsque le point mesuré se situe hors des flèches, c'est-à-dire ne correspond pas à la théorie, il sera toujours visible, au dessus ou en dessous de sa paire de flèches. Sur un écran polychrome, on pourra alors le faire apparaître comme un pixel rouge; l'expérience prouve que cette représentation colorée transmet l'information de coïncidence, parfaite, acceptable ou inacceptable, visuellement, pour l'ensemble du graphe, au cerveau humain d'une manière très efficace donc

rapide.

Mais l'expérience reste ce qu'elle est : un absolu... relatif.
Absolu, parce que si les conditions expérimentales sont reproductibles et suffisamment fines, les flèches conserveront, sur tout graphe dessiné dans les mêmes conditions, la même position de leur trou central pour tous les points. Honnêtement, et dans tous les cas, seuls les points rouges sont à prendre en considération pour affiner la seule entité maintenant modifiable... la théorie.

Relatif, parce que, si, par un moyen quelconque on parvient à réduire la hauteur des flèches, les points verts vont, petit à petit mais toujours discrètement, devenir rouges et ce, pour l'ensemble des points possibles de toutes les trajectoires possibles. L'optimiseur idéal, car le travail de l'utilisateur, économiste, physicien ou biologiste, est toujours une optimisation permanente, doit être capable de corriger F pour toutes les trajectoires possibles dans l'hyperespace des paramètres, donc de supprimer un maximum de points rouges.

On assiste donc à un match contradictoire: Plus on corrige F, moins il reste de points rouges; plus on affine l'expérience, plus il apparaît de point rouges. A ce stade, il importe d'effectuer plusieurs remarques :

Si, sur ce graphe visuel, on décide d'appliquer à chaque fois une transformation sur les ordonnées ramenant les trous des couples de flèches sur une même ligne horizontale, on perd l'information concernant les ordonnées absolues, mais on conserve la seule information nécessaire pour pratiquer l'optimisation, indépendamment du choix de la méthode d'optimisation, les écarts. Ceux-ci sont alors donnés, pour chaque point rouge par un nombre entier nécessairement non nul, exprimant une altitude, positive ou négative, en pixels.

Les suites de valeurs Yt et Ym peuvent être avantageusement remplacées par une nouvelle suite que l'on va définir maintenant, avant de choisir la méthode d'optimisation.

Ladite suite est nécessairement un masque binaire, de même longueur que les suites Yt ou Ym : appelons-la R comme rouge. Elle contient 0 si le point n'est pas rouge, donc n'intervient pas dans la future optimisation, et 1 dans le cas contraire. Numériquement, ces mêmes valeurs 0 et 1 sont aussi des coefficients de pondération réduits à leur plus simple expression : une opposition logique : soit "inutile de considérer", soit "à considérer", l'une des propositions excluant l'autre ipso facto.

Par une autre transformation conceptuelle, ramenons toutes les flèches à la même longueur (elles sont déjà à la même altitude) en affichant un point au dessus s'il est rouge sur l'écran polychrome et rien du tout s'il n'est pas rouge (peu importe

que dans le graphe original le point rouge se situe au-dessus ou en dessous des flèches). Supprimant alors la flèche du bas, nous obtenons un dessin ou plutôt un diagramme tel que



Maintenant, le choix d'une méthode d'optimisation plutôt qu'une autre, sur un critère plutôt qu'un autre, donc d'une fonction de coût quelconque, constitue toujours un arbitraire.

Seul le but final a de l'importance faire disparaître les points rouges, ramener leur ensemble à un ensemble vide.

Indépendamment du domaine d'application, de la complexité de F (sa non-linéarité), du nombre de mesures mises en jeu, des valeurs numériques mesurées ou calculées, de la trajectoire choisie pour ce faire, on a parfaitement le droit d'essayer d'optimiser la théorie sur le seul critère d'existence ou non d'un écart entre l'expérience et cette dernière, et ce, pour simplifier, en faisant appel à la méthode effectivement la plus simple et la plus connue, celle des moindres carrés.

Toutefois, avant de tenter de l'appliquer, il convient de rapporter, le plus objectivement possible, certaines critiques couramment émises à son sujet.

On a souvent reproché à la méthode des moindres carrés de ne pas considérer les écarts entre les points situés n'importe où sur les graphes de la même façon c'est-à-dire avec la même importance : la méthode optimise alors mieux les points pour lesquels les écarts sont plus grands en valeur absolue.

Déjà, on peut s'apercevoir qu'une optimisation sur la seule existence (1) ou la non-existence (0) doit être indépendante de l'ordre des points sur le graphe. On sait en effet qu'il est possible d'effectuer sur t un changement de variable tel qu'il en résultera, sur le diagramme ci-dessus, une certaine permutation de l'ordre des points qui n'aura aucune incidence sur le raisonnement suivi jusqu'à maintenant l'objection s'écroule d'elle-même, sans qu'il soit nécessaire d'écrire un seule expression mathématique à ce sujet. La simple logique binaire suffit, encore une fois.

Généralisons d'ores et déjà aussi la méthode des moindres carrés en critiquant cette fois la signification du mot "carrés".

Si Z est un vecteur, suite de valeurs toutes nulles exprimant, par définition un

idéal, l'absence totale d'écart (ou d'erreur décelable) pour un ensemble d'échantillons quelconques relevés à la meilleure précision possible sur un phénomène quelconque, Z représente l'absolu.

Soit V un autre vecteur de même longueur (ou cardinal) que Z, exprimant soit par 0, pour le même ensemble, l'absence d'écart perceptible par rapport à Z, soit par 1 qui correspond à une non-absence (donc une existence) d'écart.

La stricte application des moindres carrés va consister à minimiser, par un procédé adéquat, la somme des différences V-Z élevée au carré.

Mais Z étant partout nul, ce procédé revient à minimiser V seul élevé au carré.

Mais V, étant sur toute son étendue, égal, soit à 0 soit à 1 par définition, V au carré est toujours égal à V lui-même.

On raisonnera maintenant sur V, élevé, terme à terme, à une puissance quelconque mais <u>non nulle</u>, sachant que 0 élevé à ladite puissance vaut toujours 0 , et que 1 élevé à la même puissance vaut toujours 1.

On se rend compte que ce raisonnement exprime dans toute sa généralité un théorème essentiel de l'optimisation idéale :

Toute optimisation réalisée sur un système ou ensemble quelconque exprimé par des parités d'existence ou de non-existence d'écarts observables, est, a priori, optimale, car elle ne dépend ni de l'ordre choisi pour exprimer ces écarts, ni de la puissance à laquelle on va opérer.

Ce théorème a plusieurs conséquences ou corollaires

a) Ni la théorie des Nombres ni celle des Fonctions n'interviennent plus dans le raisonnement, ni par un choix d'ordre (au sens de l'ordre dans un ensemble dit ordonné), ni par un choix d'ordre (au sens d'élévation à une puissance imposée ou arbitrairement décidée).

b) Le même raisonnement s'applique aussi si l'on désire optimiser non plus une fonction F quelconque, mais l'une quelconque de ses intégrale ou de ses dérivées, par rapport à n'importe quelle variable ou paramètre quelconque, par rapport à une combinaison quelconque des ces variables ou paramètres. Il s'appliquerait aussi dans le cas d'une intégration ou d'une dérivation d'ordre non entier, donc quelconque, couvrant ainsi, toujours a priori, l'ensemble des considérations relatives à la combinatoire ou la complexité. Le mot "ordre" devient le plus important du raisonnement, car il couvre maintenant, en outre, aussi l'ordre de dérivation et d'intégration.

Une optimisation basée sur des parités d'existence ou de nonexistence et strictement sur ces seules parités, peut donc s'effectuer dans le plus grand désordre apparent et rester pourtant optimale à tous les ordres à la fois, quelle que soit la signification donnée au mot "ordre" parmi les trois différentes considérées jusqu'ici. Il est d'ailleurs possible (mais non nécessaire) de démontrer, par un développement limité en série de Taylor ou de Mac Laurin, que les deux derniers sens du mot "ordre" (élévation à une puissance quelconque, et ordre d'intégration ou de dérivation) ne sont pas indépendants, indépendamment de la fonction F et de son domaine d'application.

L'axiomatique développée ici ne sera jamais soumise, même pour un nombre a priori infini de parités, aux conséquences du Théorème de Gödel (1931), lequel énonce que toute axiomatique aboutit un jour à tomber sur une proposition indécidable. Le théorème s'applique en effet à toute axiomatique basée sur la théorie des nombres. Or, il est lui-même démontré en admettant comme un axiome ladite théorie et devient indémontrable si l'on renonce à cet axiome.
L'axiomatique développée ici ne sera jamais soumise, même pour un nombre infini de parités, à l'axiome de continuité, toute prise de mesure par échantillonnage et tout procédé de calcul réalisable (algorithme) étant nécessairement une suite d'opérations discrètes. En outre, et par définition, il ne peut exister de continuité entre une existence et une non-existence et réciproquement : L'optimiseur idéal ne pourra donc jamais être décrit correctement par aucune fonction continue. Par contre, il pourrait être descriptible par un algorithme exprimé nécessairement dans la même algèbre que celle qui décrit déjà, encore une fois, nécessairement, les données à traiter.

Questions : A partir de ce raisonnement, et de ce raisonnement seul, peut-on prévoir :

a) si l'algorithme discret exprimant le fonctionnement de l'algorithme idéal d'optimisation sera unique, à égalité de propriétés;

b) s'il est possible d'en imaginer ou d'en trouver un encore supérieur donc plus général.

Il semble bien que la seule réponse possible à ces deux questions, à choix binaire, soit **NON**.

Déjà, et sans que l'on ait cherché à définir avec précision l'optimiseur idéal, le raisonnement ci-dessus, sauf s'il est démenti, soit par preuve du contraire (encore un choix binaire), soit par une preuve expérimentale, montre que toute tentative d'optimisation de quoi que ce soit, utilisant soit la théorie des fonctions continues soit la théorie des nombres, revêtira un caractère d'optimalité non satistaisant.

Il en serait de même de toute tentative d'infirmer le présent raisonnement, soit en utilisant une axiomatique basée sur la théorie du continu, soit une autre, même discrète, mais basée sur la théorie des nombres et sur la nécessité d'introduire la notion d'ensemble ordonné (attention à Gödel), soit, ce qui serait encore pire, sur une combinaison des deux axiomes inutiles contestés ici (cas de la théorie des fonctions continues). La seule infirmation possible ne peut provenir que d'un autre raisonnement exclusivement logique (ne comportant aucun nouveau postulat), ou, pire, que d'une illogique basée sur l'arbitraire, donc sur le hasard, ou sur la poursuite obstinée d'une croyance forte en certaines erreurs initiales.

(Toute théorie se comporte comme un système dynamique : quelle que soit la fonction ou l'algorithme qui la font évoluer, de préférence dans le bon sens, elle restera essentiellement soumise à ses conditions initiales, renfermées dans son axiomatique initiale, donc dans ses postulats.)

En outre, l'optimiseur idéal, basé sur la seule logique des parités (ou des contraires) deviendrait à la fois l'énoncé correct (et le seul possible) à la fois du Principe de Moindre Action sous sa forme primitive la plus générale, indépendante de tout postulat et de toute grandeur de nature macroscopique ("La Nature est économe dans toutes ses actions", Maupertuis), et aussi celui de l'interaction élémentaire, si elle existe, comme les physiciens le postulent et la recherchent à grand renfort de crédits.

Question : Compte tenu du raisonnement ci-dessus, supposé irréfutable, (mais qui sait?), quelle est la seule formulation possible pour le principe de moindre action généralisée, l'optimiseur idéal, donc l'interaction élémentaire, à la fois ?

(Réponse dans le prochain numéro).

Note du rédacteur : il semble qu'il n'y ait pas de prochain numéro...ou que la réponse n'ait pas été donnée.

CALCUL DE RACINES EN « J »

R. Coquidé

Racines d'un polynôme

(expression usuellement utilisée pour « solutions d'une équation polynomiale »)

Le langage J est pourvu d'un outil puissant pour calculer les racines d'un polynôme : le verbe « p. ».

Il est possible de définir un polynôme par ses racines.

Par exemple le polynôme suivant défini par ses facteurs

```
3(x-1)(x-2)(x+1.5)(x-0.5)(x-1.2)
```

sera créé par

```
[pr =. 3;1 2 _1.5 0.5 1.2      NB. On peut remarquer le
┌─┬────────────────────┐       NB. facteur 3 et la «mise en boîte »
│3│1 2 _1.5 0.5 1.2    │       NB. de la variable pr (pro-nom)
└─┴────────────────────┘
```

Avec le verbe « p. », calcul des coefficients du polynôme :

```
   [pc =. p. pr                NB. A partir des racines
5.4 _19.8 19.05 1.95 _9.6 3    NB. Ici, pas de « boîte »
```

Avec le même verbe « p. » calcul des racines à partir des coefficients :

```
    p. pc       NB. rangées dans l'ordre des modules décroissants
┌─┬──────────────────┐
│3│2 _1.5 1.2 1 0.5  │
└─┴──────────────────┘
```

Les racines complexes sont acceptées ... et traitées...

```
   [pr =. 2 ; 0j0.2 _1 0j_0.2 1j2 1j_2 2
┌─┬────────────────────────────────┐
│2│0j0.2 _1 0j_0.2 1j2 1j_2 2      │
└─┴────────────────────────────────┘
   [pc =. p. pr
_0.8 _0.08 _19.6 _2.24 10.08 _6 2
   p. pc
┌─┬────────────────────────────────┐
│2│1j2 1j_2 2 _1 0j0.2 0j_0.2      │
└─┴────────────────────────────────┘
```

... les coefficients complexes aussi

```
   [pr =. 2j_1 ; 1j0.1 _1 0j_0.2 1j2 1j_2 _2j_0.5
┌────┬──────────────────────────────────────┐
│2j_1│1j0.1 _1 0j_0.2 1j2 1j_2 _2j_0.5      │
└────┴──────────────────────────────────────┘
   [pc =. p. pr
_0.55j_4.6 _23.13j2.39 _0.57j5.01 22.25j_6.35 _1.16j_2.17
  0.6j1.2 2j_1
   p. pc
┌────┬──────────────────────────────────────────┐
│2j_1│1j2 1j_2 _2j_0.5 1j0.1 _1j0 0j_0.2        │
└────┴──────────────────────────────────────────┘
```

Solution de f(x) = 0 où f(x) n'est pas polynomiale

(on suppose connu un intervalle [a , b] contenant une solution et une seule : x).

Il est permis de s'inspirer de l'algorithme de NEWTON (ou de la tangente):

$x_{n+1} = x_n - f(x_n)/f'(x_n)$ qui calcule x_{n+1}, approximation meilleure que x_n

NEWTON =.]-([. %].) NB. conjonction

Usage : x_{n+1} =. (f NEWTON fp) x_n

x_{n+k} =. ((f NEWTON fp)^: k) x_n

Ici, on utilise la « puissance fonctionnelle » k (entier positif ou nul).

La suite récurrente (x_n) converge (quand tout se passe bien !) vers x, racine de l'équation f(x) = 0 ; fp(x) est la dérivée de f(x)

Exemple : pour résoudre x = cos(x) ou f(x) = 0 avec f(x) = x – cos(x), écrivons :

```
 f =. -2&o.        NB. verbe représentant la fonction x-cos(x)
fp =. 1: + 1&o.    NB. verbe représentant sa dérivée première
```

Partons de x_0 = 1.5

```
NB. Si nous utilisons
NB. xn+1 = cos(xn)
,. (2&o.)^:(i.20) 1.5
1.5
0.070737201667703
0.997499167206586
0.542404992339220
0.856469708947328
0.655108801780784
0.792981645797353
0.701724168276573
0.763730311290859
0.722261082134520
0.750312885709986
0.731475558034674
0.744189586630583
0.735637098348715
0.741403372909269
0.737521554461599
0.740137474793432
0.738375854160239
0.739562726170268
0.738763336576956
etc...
```

Convergence lente
(d'ordre 1)

```
NB. Si nous utilisons
NB. la conjonction NEWTON
,. (f NEWTON fp)^:(i.6) 1.5
1.5
0.784472397719411
0.739518709832052
0.739085174705196
0.739085133215161
0.739085133215161
```

Convergence beaucoup
plus rapide
(d'ordre 2).

On a défini et exécuté

```
P =. 9!:11
P 15
```

pour obtenir 15 chiffres affichés.

```
,. (f NEWTON fp)^:(_) 1.5
0.739085133215161
```

Utilisation de la « puissance fonctionnelle » infinie.

A suivre!

Développement Objet en APL+Win 3.5

par Eric Lescasse

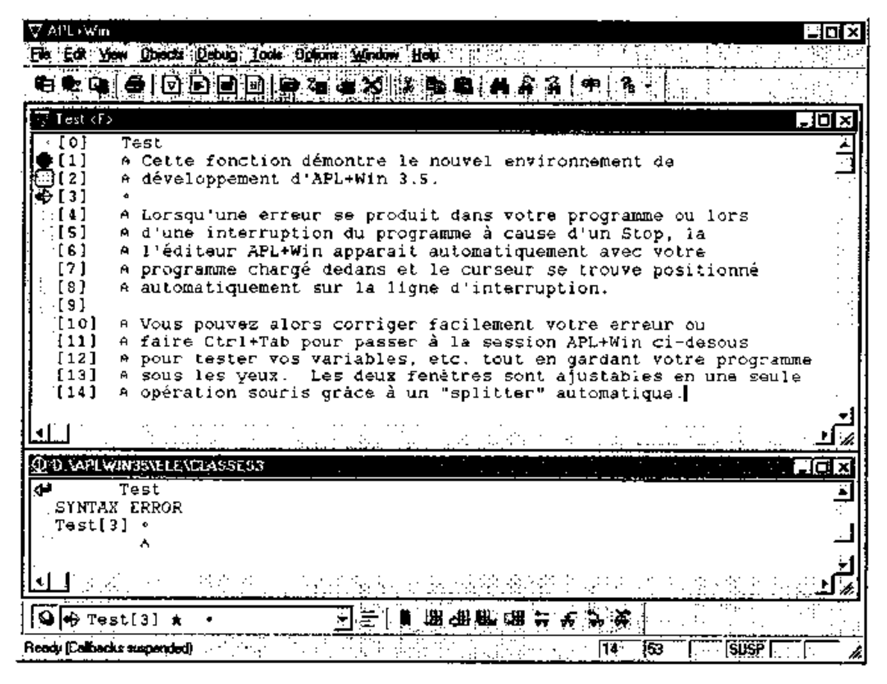
Introduction

La version **3.5** d'**APL+Win** va sortir dans quelques semaines et pour la première fois, il sera possible de créer ses propres objets en **APL+Win**, au **niveau système**.

Le but de cet article est de décrire cette nouvelle fonctionnalité, l'une des plus importantes innovations **APL** depuis de nombreuses années.

APL+Win 3.5

Avant tout, décrivons succinctement quelques nouvelles caractéristiques d'APL+Win 3.5 **dont nous nous servirons dans cet article.**



Cet image écran démontre le nouvel environnement de développement d'**APL+Win 3.5**.

Vous pouvez remarquer :

- Le nouveau « look » à base de boutons plats de type Word 97
- La présence d'une **barre d'outils « debogueur »** au-dessus de la barre de statut
- La présence d'une nouvelle « gouttière » le long de l'éditeur **APL+Win** et de la session **APL+Win**
- L'ouverture automatique de l'éditeur **APL+Win** avec positionnement du curseur sur la ligne fautive lors d'une interruption de programme (erreur ou « stop »)

Vous pouvez passer de l'éditeur à la session APL, par l'appui de **Ctrl+Tab.**

Cet environnement de développement est idéal pour mettre au point ses programmes.

La gouttière sert à placer/retirer visuellement des « stop » , des « trace » et des marqueurs (« **bookmarks** »).

Les marqueurs sont très utiles lorsque vous travaillez sur vos objets. En effet, comme nous allons le voir dans cet article, les objets sont souvent des fonctions APL de grande tailles, puisqu'ils doivent incorporer de nombreux comportements et propriétés. Une fois quelques marqueurs positionnés par l'appui de **Ctrl+F2** sur la ligne comportant le curseur, il vous suffit d'appuyer sur **F2** pour sauter d'un marqueur à l'autre. La navigation dans un très long programme devient très aisée et rapide.

Par ailleurs citons quelques autres améliorations de l'environnement de développement **APL+Win 3.5:**

- La limite maximum du nombre de lignes mémorisées dans la session **APL** est portée de **2000** à **8000** lignes
- Des bulles explicative apparaissent automatiquement lorsque le curseur survole un nom de fonction ou de variable dans la session **APL** ou l'éditeur[1]
- Le support des noms longs d'espace de travail
- Des barres d'outils personnalisables et flottantes/détachables
- F1 ouvre l'aide contextuelle correspondant au mot sous lequel se trouve le curseur
- ⎕*STOP* et ⎕*TRACE* positionnables visuellement par Ctrl+. Et Ctrl+, dans l'éditeur

Outre les améliorations de l'environnement de développement **APL** (une partie négligée par tous les fournisseurs **APL** depuis de longues années), **APL+Win** 3.5 comporte un très grand nombre d'améliorations touchant tous les aspects de l'APL. Il serait trop long et hors de propos de les décrire ici.

Intéressons-nous maintenant à l'amélioration la plus importante : la possibilité de définir et d'utiliser ses propres objets.

[1] Par exemple, lorsque le curseur survole le nom de la variable *aaa* définie comme *aaa*←⍳10, la bulle fait apparaître : *aaa* ← 10 ⍴ 1 2 3 4 5 6 7 8 9 10

Rappel sur la programmation objet en APL+Win

Avant d'analyser comment **APL+Win** vous permet de créer puis utiliser vos propres classes d'objets, il est utile de rappeler comment fonctionne la programmation objet en **APL+Win**.

La programmation objet en **APL+Win** s'effectue à l'aide des fonctions systèmes ⎕*WI* et ⎕*NI*.

Les objets que vous pouvez utiliser en **APL+Win** sont en standard:

Tous les objets de Windows 95/98 et NT (**Button, Check, Combo, Edit, Form, Frame, Imagelist, Label, List, Listview, MDIForm, Media, Menu, Option, Page, Picture, Printer, Progress, RichEdit, Scroll, Selector, Spinner, Status, Timer, Toolbox, Trackbar, Tree** et l'objet # ou objet Système)

Tous les objets **OCX** et **ActiveX** existants (des milliers d'objets, Freeware, Shareware et produits commerciaux)

Tous les logiciels qui supportent la technologie **COM** de **Microsoft**, tels **Excel**, **Word**, etc.

Les objets de communication **TCP/IP** (avec la fonction système ⎕*NI*)

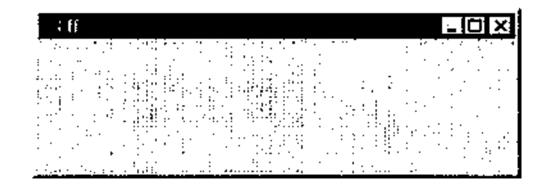
Tous les objets que vous pouvez utiliser en APL+Win se caractérisent par:

Des **propriétés** (=des états de ces objets)
Des **méthodes** (=des actions que l'on fait effectuer à ces objets)
Des **événements** (=des informations que ces objets transmettent lorsqu'ils subissent des actions)

Quelques exemples d'utilisation d'objets en APL+Win

Créons une fenêtre:

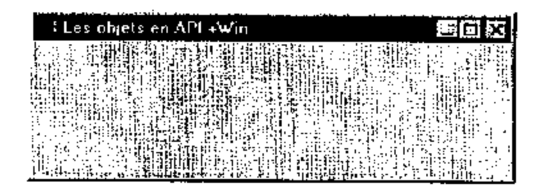
```
      'ff' ⎕WI 'New' 'Form'
ff
```



Par cette instruction nous créons une fenêtre Windows. En fait nous créons une nouvelle (**New**) instance de la classe d'objet fenêtres (**Form**) et nous donnons le nom **ff** à cette instance. Nous avons utilisé un premier objet: l'objet **Form**.

Changeons deux propriétés de cet objet:

```
      'ff' ⎕WI 'caption' 'Les objets en APL+Win'
```



Par cette instruction nous indiquons que nous voulons modifier le titre (**caption**) de notre instance de fenêtre (que nous appellerons désormais abusivement "notre objet") **ff**.

Nous pouvons de même interroger la taille de la fenêtre (sa propriété **size**), puis modifier cette taille et modifier la couleur de la fenêtre (propriété **color**).

```
      'ff' ⎕WI 'size'
5.9375 42.375

      'ff' ⎕WI 'size' 12.25 28

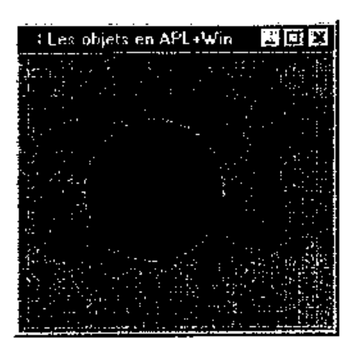
      'ff' ⎕WI 'color' 0 255 0
```



Maintenant, rendons notre fenêtre "toujours visible" puis appliquons une méthode à notre fenêtre.

```
      'ff'⎕WI'style'16

      'ff'⎕WI'Draw' 'Clear'('Brush'1 255)('Circle'6 12 6 .5)
```



Finalement, faisons en sorte de réagir à un événement se produisant sur notre fenêtre. Les événements possibles sont:

```
      'ff' ⎕WI 'events'
Close DdeConnect DdeDisconnect DdeExecute Delete Destroy DragEnter
   DragLeave DragOver Drop DropDown ExitError Focus Hide KeyDown
   KeyPress KeyUp Modified MouseDouble MouseDown MouseDrag  MouseE
   nter MouseLeave MouseMove MouseUp Move Open Paint Reopen Resize
    Send Show Unfocus Wait
```

Indiquons à **APL+Win** que nous souhaitons être informé de tout **clic souris** se produisant dans notre fenêtre.

```
      'ff' ⎕WI 'onMouseDown' '⎕WARG'
```

Cette instruction indique à l'interpréteur **APL+Win**, que dès qu'un clic (événement **MouseDown**) se produira dans le fenêtre, et à ce moment seulement, il devra exécuter la chaîne de caractères et donc afficher dans la session APL, le contenu de la variable système . Cette variable contient toujours les informations importantes relatives à l'événement qui vient de se produire.

Maintenant, voyons si nous sommes précis et essayons de cliquer au centre du cercle rouge. Je viens de le faire et voici ce qui s'est affiché dans la session APL:

```
6 12.125 1 1 0
```

Mon clic est tombé assez proche du centre du cercle dont les coordonnées étaient **6 12** (voir l'instruction **Draw ... Circle** ci-dessus)!

Pour l'instant nous n'avons utilisé qu'un objet de class **Form**.

Utilisation d'un objet ActiveX

Utilisons maintenant un objet **ActiveX**, c'est à dire un objet externe à **APL+Win**.

La grande nouvelle est que vous disposez, peut être sans le savoir, de dizaines, voire de centaines de ces objets, gratuitement, sur votre propre micro ordinateur. Et certains d'entre eux sont exceptionnels de fonctionnalités et d'utilité! Et vous pouvez vous en servir aussi

facilement en **APL+Win** que s'il s'agissait d'un objet interne à **APL**, avec la même fonction et la même syntaxe qui a déjà été exposée ci-dessus.

Comment connaître la liste de tous ces merveilleux objets?

C'est très simple: il suffit d'interroger la propriété '**xclasses**' de l'objet système:

```
      ⍴aaa←'#'⎕wi'xclasses'
259 4
```

Comme vous le voyez, je dispose personnellement de **259** objets **ActiveX** sur mon ordinateur. Je n'en liste ci-dessous que quelques uns au hasard. Ces objets sont très variés. Avec les quelques objets cités, je peux facilement réaliser des applications **APL+Win** qui:

- Gèrent des images
- Utilisent des objets grilles sophistiqués (**Formula One** et **Videosoft**)
- Incorporent **Internet Explorer 5** dans mes fenêtres APL!!!
- Font de superbes graphiques en 2D et 3D (**ChartFX**)
- Contrôlent l'orthographe de documents (**VisualSpeller**)
- Transfèrent des fichiers sur Internet (**Microsoft Internet Transfer Control**)
- Utilisent des champs texte avec masques de saisie dans mes fenêtres
- Et même créent un véritable éditeur **HTML WYSIWYG** de haute qualité grâce à l'objet **DHTML Edit Control de Microsoft**.

```
      aaa[;1 2]
Contr⍒le d'⊖dition d'images Kodak           {6D940280-9F11-11CE-83FD-02608C3EC08A}
Microsoft Forms 2.0 ToggleButton            {8BD21D60-EC42-11CE-9E0D-00AA006002F3}
Navigateur Web Microsoft                    {8856F961-340A-11D0-A96B-00C04FD705A2}
VC Formula One 5.0 Workbook                 {13E51003-A52B-11D0-86DA-00608CB9FBFB}
ChartFX Control                             {8996B0A1-D7BE-101B-8650-00AA003A5593}
VisualSpeller Control                       {97F4CED0-9103-11CE-8385-524153480001}
:-) VideoSoft FlexGrid Control              {6099FCD0-0A81-11D2-BAA4-04F205C10000}
Microsoft Internet Transfer Control, version 6.0 {48E59293-9880-11CF-9754-00AA00
C00908}
Microsoft Masked Edit Control, version 6.0  {C932BA85-4374-101B-A56C-00AA003668D
C}
DHTML Edit Control for IE5                                    {2D360200-FFF5-11d1
-8D03-00A0C959BC0A}
DHTML Edit Control Safe for Scripting for IE5                 {2D360201-FFF5-11d1
-8D03-00A0C959BC0A}
```

Et je n'ai listé là qu'une dizaine d'objets parmi les **259** dont je dispose.

Un exemple:

Créons une fenêtre **APL** contenant **Internet Explorer 5**, connectons-nous à Internet et allons visiter mon site Web:

```
'int' ⎕WI 'New' 'Form'
'int.msie5' ⎕WI 'New' 'Navigateur Web Microsoft'
'int'⎕WI'onResize' "'int.msie5'⎕WI'where'0 0,'int'⎕WI'size'"
```

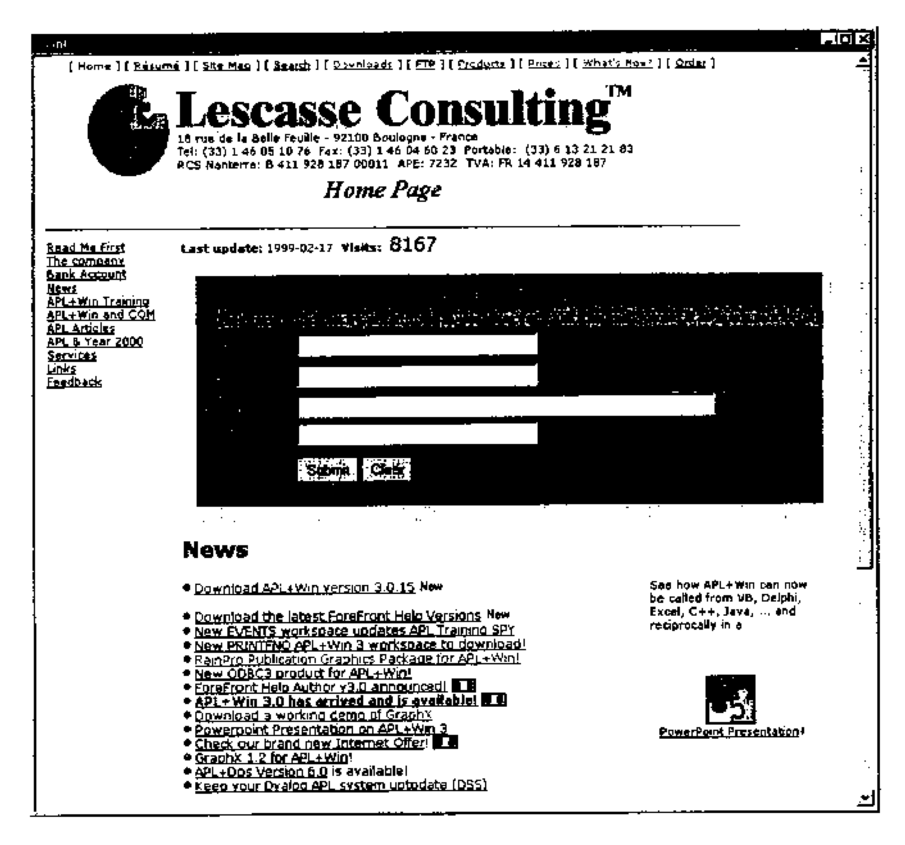
Le première instruction crée une fenêtre **APL**.

La seconde crée une instance du navigateur **Internet Explorer 5** et l'incorpore dans le fenêtre (la notation **int.msie5** indique que l'objet **msie5** est un enfant de la fenêtre **int**).

Vous noterez que **Navigateur Web Microsoft** est le nom exact de l'un des objets **ActiveX** existants sur ma machine (voir ci-dessus)
La troisième instruction indique à **APL+Win** qu'il devra redimensionner **Internet Explorer** en lui donnant les dimensions exactes de la fenêtre chaque fois que celle-ci changera de taille (c'est à dire lorsque je modifierai la taille de la fenêtre à l'aide de la souris)

Maintenant, supposons que j'active ma connexion Internet. Pour aller visiter mon site Web, avec ma mini application APL, il suffit que j'écrive:

```
'int.msie5' ⎕WI 'Navigate' 'http://www.lescasse.com'
```



Il n'est bien sûr pas nécessaire que **Internet Explorer** occupe la totalité de ma fenêtre **APL** et je pourrais aisément créer une application **APL** sophistiquée qui utilise **Internet Explorer** dans une partie de la fenêtre de mon application!

Bien sûr, la fenêtre ci-dessus n'est pas seulement un image mais bien une instance complète **d'Internet Explorer 5**, me permettant de naviguer librement sur mon site et sur **Internet**, en cliquant sur les liens hypertextes qui apparaissent dans la page.

Au passage, j'en profite pour vous indiquer qu'APL **(aussi bien** APL+Win **que** Dyalog APL**) permet désormais de développer des applications** Internet **complètes et complexes. Il est en effet possible de développer un** Serveur Web **en** APL **(en moins de** 200 **lignes d'**APL+Win**).**

Si vous vous connectez sur mon site **Web** à l'adresse **http://www.lescasse.com** (voir ci-dessus) entrez votre nom et votre adresse **E-mail** dans le formulaire **Guest Book**, qui se présente à vous. Vous recevrez alors une page de réponse qui vous expliquera qu'un **Serveur Web APL** vient de traiter votre requête et vous présentera quelques citations très intéressantes de développeurs américains concernant **APL+Win** et **Internet**.

Lorsque vous cliquerez sur le bouton **Submit**, ces informations seront automatiquement envoyées à mon serveur Internet dans le centre de Paris et traitées sur ce serveur par un **APL+Win**.

Ce **Serveur Web APL** est un "**Service NT APL+Win**" qui fonctionne de façon permanente et fiable depuis plusieurs mois maintenant.

Il analyse les réponses du formulaire, les enregistre dans une base de données (un simple fichier **APL** à composantes) et m'envoie automatiquement un **E-mail** pour me prévenir et m'informer du visiteur qui vient d'aller voir mon site.

Pour l'instant, mon Serveur Web **APL** ne gère que ce formulaire, mais il a été construit pour gérer autant de formulaires, compteurs Internet, bases de données, etc. qu'il est possible d'imaginer et ceci pour tous les visiteurs simultanés qui se connectent sur mon site.

Utilisation d'objets COM

Un autre type de développement objet consiste à exploiter la technologie **COM** de **Microsoft**.

De façon très simplifiée cette technologie a été créée pour permettre à des objets et logiciels disparates de communiquer et d'échanger des données entre eux.

A titre d'exemple, montrons comment il est possible en quelques instructions **APL** de complètement piloter **Excel** et d'exploiter **Excel** depuis une application **APL+Win**.

L'expression:

```
'ex' ⎕wi 'New' 'Excel.Application'
```

démarre **Excel 97** sous la forme d'un objet **APL** nommé **ex** et le charge en mémoire (sans le montrer pour l'instant).

Vous pouvez immédiatement demander à voir la liste de propriétés, méthodes et événements disponibles, comme pour tout objet. Ces listes étant trop longues contentons-nous de demander le nombre de propriétés, méthodes et événements disponibles pour notre objet **Excel**:

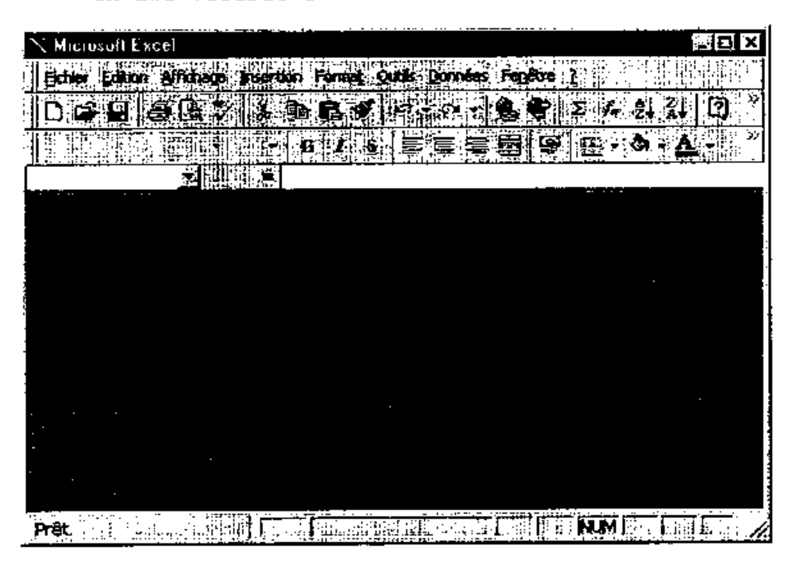
```
⍴'ex'⎕wi'properties'
```

```
159
      ρ'ex'⎕wi'methods'
60
      ρ'ex'⎕wi'events'
6
```

Les objets **COM** sont en réalité constitués d'une hiérarchie d'objets et sous-objets. L'accès aux sous-objets se fait par l'intermédiaire de certaines propriétés de l'objet **COM**, avec une syntaxe de "redirection" de cette propriété vers un sous objet.

Commençons par montrer notre objet **Excel**:

```
      'ex'⎕wi'visible'1
```



Vous remarquerez qu'aucun classeur n'est ouvert par défaut. En effet un classeur **Excel** est un objet (en fait un sous-objet **d'Excel**) et il nous faut le créer.

Il faut tout d'abord créer un objet **"Collection de Classeurs"**. Appelons-le **ex.wkbks**:

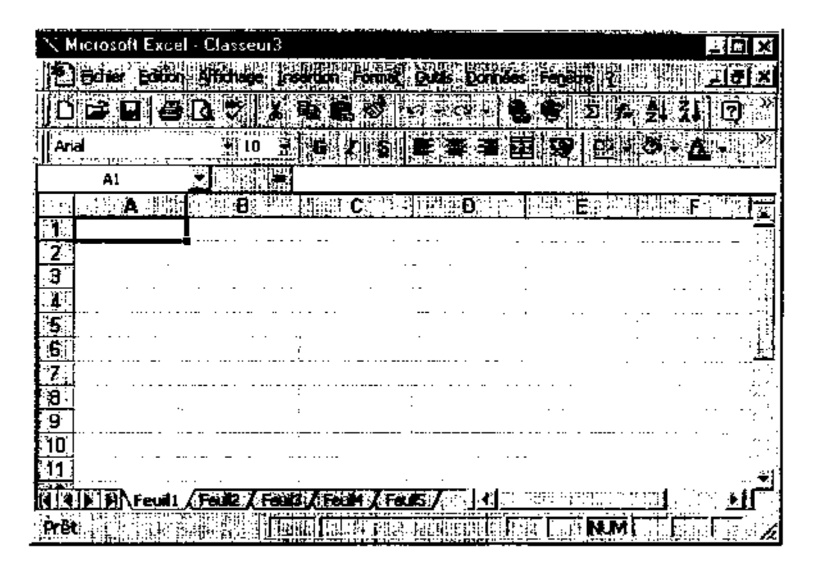
```
      'ex'⎕wi'Workbooks>ex.wkbks'
```

La syntaxe précédente, nouvelle pour ⎕wi, permet de rediriger la propriété **Workbooks** vers la création d'un objet relatif à cette propriété.

Maintenant, créons un classeur (=ajoutons un classeur **Excel** à notre collection), avec **5** feuilles de calcul:

```
      'ex'⎕wi'SheetsInNewWorkbook'5
 'ex.wkbks'⎕wi'XAdd>ex.wkbk'
```

Notre fenêtre **Excel** devient:



Il nous faut maintenant rendre actif le **Classeur** ainsi créé.

```
'ex.wkbks'⎕wi'Item>ex.wkbk'1
  'ex.wkbk'⎕wi'Activate'
```

La documentation du modèle objet **d'Excel** nous indique qu'il faut maintenant créer un objet "**collection de Feuilles**" ainsi qu'un objet "**Feuille**" et rendre actif cet objet feuille en passant son numéro à la propriété **Item**, puis en invoquant la méthode **Activate**:

```
  'ex.wkbk'⎕wi'Worksheets>ex.wkshts'
'ex.wkshts'⎕wi'Item>ex.wksht'2
 'ex.wksht'⎕wi'Activate'
```

Ces 3 instructions sont similaires aux instructions ayant permis de créer les objets "**collection de Classeurs**" et "**Classeur**" ci-dessus.

Par exemple, créons un tableau généralisé en APL:

```
      report←?5 3ρ10000
      report←'APL+Win' 'APL+Dos' 'APL+Unix' 'APL+PC' 'APL+Li
nk',report
      report←(AV2ANSI¨'' 'Jan' 'Fev' 'Mar')⍪report
      report
           Jan  Fev  Mar
 APL+Win  1742 6851 5158
 APL+Dos  9066 5967 4005
 APL+Unix 7432 9805 3459
 APL+PC   3804 9630 8291
 APL+Link  996 1328 1045

      ⍴report
6 4
```

Pour installer ce tableau dans la feuille **Excel** active, il faut créer un objet **"plage de cellules"** ("**range**" en anglais), enfant de notre feuille:

```
     'ex.wksht'⎕wi'Range>ex.rng' 'B2:E7'
       'ex.rng'⎕wi'Value'(⍉report)
```

Notez qu'**Excel** traite ses tableaux en plaçant les colonnes avant les lignes, contrairement à **APL**, d'où la nécessité de transposer notre tableau généralisé **APL**.

Ajoutons maintenant une série de formules en bas du tableau pour effectuer les sommes des colonnes:

```
     'ex.wksht'⎕wi'Range>ex.rng' 'C8:E8'
       'ex.rng'⎕wi'Formula' '=sum(C3:C7)'
```

Notez que nous n'avons indiqué qu'une formule pour 3 cellules et qu'Excel s'est chargé tout seul, d'une part de reproduire la formule dans chacune des 3 cellules **C8, D8** et **E8** et d'autre part de la traduire en **=SOMME(C3:C7) =SOMME(D3:D7)** et **=SOMME(E3:E7)**.

Maintenant encadrons les diverses plages de titre:

```
      'ex.wksht'⎕wi'Range>ex.rng' 'B3:B7'
        'ex.rng'⎕wi'BorderAround'1 3
      'ex.wksht'⎕wi'Range>ex.rng' 'B2:E2'
        'ex.rng'⎕wi'BorderAround'1 3
      'ex.wksht'⎕wi'Range>ex.rng' 'B8:E8'
        'ex.rng'⎕wi'BorderAround'1 3
      'ex.wksht'⎕wi'Range>ex.rng' 'B2:E8'
        'ex.rng'⎕wi'BorderAround'1 4
```

Puis alignons à droite le contenu des colonnes **C D** et **E**:

```
    'ex.wksht'⎕wi'Range>ex.rng' 'C2:E8'
      'ex.rng'⎕wi'HorizontalAlignment' ¯4152
```

Changeons maintenant la police des titres de ligne et de colonnes de notre tableau. Pour cela il faut créer un objet "**police**" qui soit un enfant de l'objet "**plage**":

```
    'ex.wksht'⎕wi'Range>ex.rng' 'B3:B7;C2:E2;B8:E8'
      'ex.rng'⎕wi'Font>ex.rng.fnt'
  'ex.rng.fnt'⎕wi'xName' 'Arial'
  'ex.rng.fnt'⎕wi'size'12
  'ex.rng.fnt'⎕wi'Bold'1
```

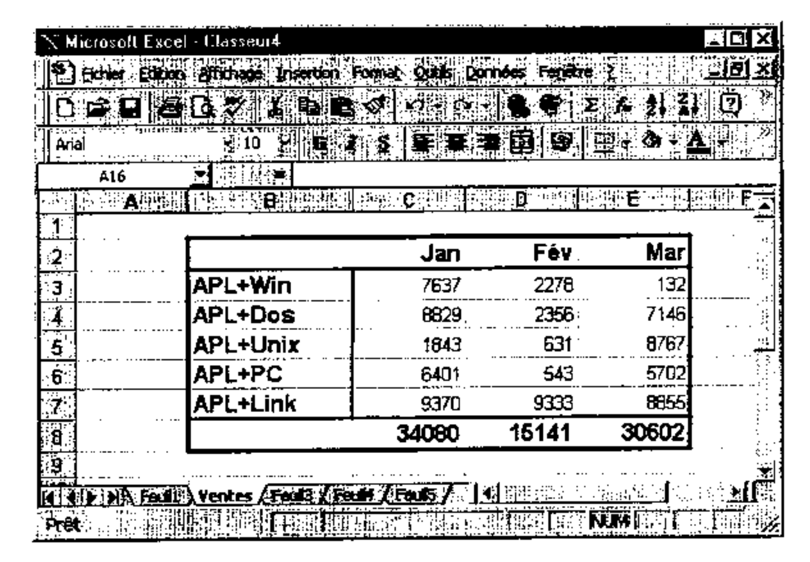
Puis élargissons la colonne **B** pour tenir compte de la police de plus grande taille:

```
    'ex.wksht'⎕wi'Range>ex.rng' 'B1'
      'ex.rng'⎕wi'ColumnWidth'16
```

Pour terminer donnons le nom "**Ventes**" à l'onglet de la feuille active:

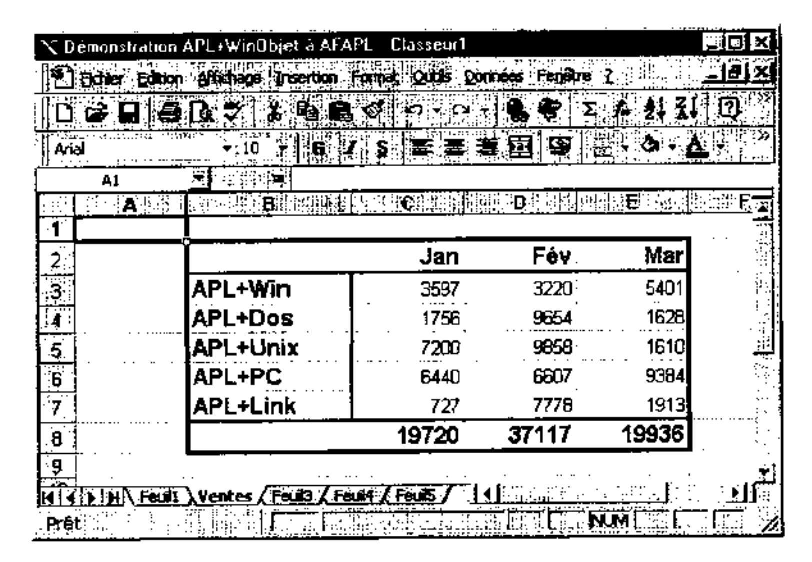
```
    'ex.wksht'⎕wi'xName' 'Ventes'
```

Finalement notre objet **Excel** et notre tableau ont l'aspect suivant:



Changeons le titre de notre fenêtre **Excel**:

'ex'⎕wi'caption'(AV2ANSI'Démonstration APL+Win Objet à AFAPL')



Notez l'utilisation de la fonction utilitaire *AV2ANSI* (livrée avec **APL+Win**) pour convertir les caractères du ⎕*AV* en caractères **ANSI**.

Il faut bien que grâce à la technologie **COM, Excel** est devenu un objet **qu'APL** peut manipuler à loisir, exactement comme nous le souhaitons.

Enfin, si nous souhaitions sauvegarder notre **workbook**, rien ne serait plus simple:

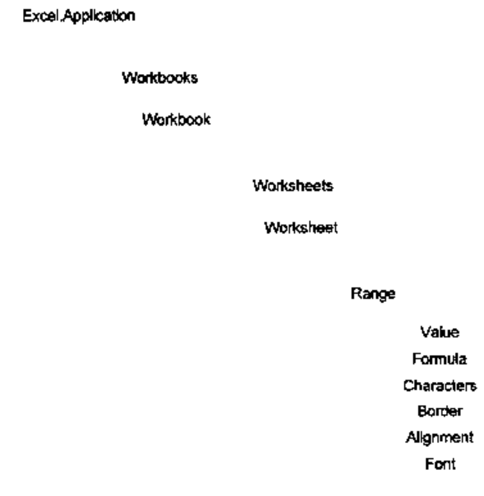
```
'ex.wkbk'⎕wi'SaveAs' 'c:\temp\afapl.xls'
```

et pour le recharger plus tard:

```
'ex.wkbks'⎕wi'XOpen' 'c:\temp\temp.xls'
```

Qu'avons nous appris à l'étude de cet exemple?

1. Des produits tels **qu'Excel, Word,** etc. sont des objets **ActiveX** qu'il est possible de manipuler entièrement depuis APL+Win
2. La syntaxe à utiliser ne fait appel qu'à une seule fonction système **APL+Win** (⎕*wi*) et est très simple
3. Un produit comme **Excel** est une hiérarchie d'objets que l'on peut représenter ainsi:



Résumons toutes les instructions APL **relatives à notre exemple** Excel **dans une fonction** APL:

```
    ∇ ExcelExample;report
      ⍝∇ ExcelExample -- Démontre le pilotage d'Excel depuis APL
      ⍝∇ avec la technologie COM

      report←?5 3⍴10000
      report←'APL+Win' 'APL+Dos' 'APL+Unix' 'APL+PC' 'APL+Link',report
      report←(AV2ANSI¨'' 'Jan' 'Fév' 'Mar')⍪report

      ⎕wself← 'ex'⎕wi'Create' 'Excel.Application'('visible'1)
              'ex'⎕wi'Workbooks>ex.wkbks'   ⍝ crée un nouveau classeur
              'ex'⎕wi'SheetsInNewWorkbook'5 ⍝ nombre de feuilles à ajouter
        'ex.wkbks'⎕wi'XAdd>ex.wkbk'
        'ex.wkbks'⎕wi'Item>ex.wkbk'1
         'ex.wkbk'⎕wi'Activate'
         'ex.wkbk'⎕wi'Worksheets>ex.wkshts'
       'ex.wkshts'⎕wi'Item>ex.wksht'2
        'ex.wksht'⎕wi'Activate'

        'ex.wksht'⎕wi'Range>ex.rng' 'B2:E7' ⍝ envoi d'un tableau APL ds Excel
          'ex.rng'⎕wi'Value'(⍉report)

        'ex.wksht'⎕wi'Range>ex.rng' 'C8:E8' ⍝ création de formules ds Excel
          'ex.rng'⎕wi'Formula' '=sum(C3:C7)'

        'ex.wksht'⎕wi'Range>ex.rng' 'C2:E8' ⍝ changement alignement horizontal
          'ex.rng'⎕wi'HorizontalAlignment' ¯4152

        'ex.wksht'⎕wi'Range>ex.rng' 'B3:B7' ⍝ encadrement de plages de cellules
          'ex.rng'⎕wi'BorderAround'1 3
        'ex.wksht'⎕wi'Range>ex.rng' 'B2:E2'
          'ex.rng'⎕wi'BorderAround'1 3
        'ex.wksht'⎕wi'Range>ex.rng' 'B8:E8'
          'ex.rng'⎕wi'BorderAround'1 3
        'ex.wksht'⎕wi'Range>ex.rng' 'B2:E8'
          'ex.rng'⎕wi'BorderAround'1 4

        'ex.wksht'⎕wi'Range>ex.rng' 'B3:B7;C2:E2;B8:E8' ⍝ changement de police
          'ex.rng'⎕wi'Font>ex.rng.fnt'
      'ex.rng.fnt'⎕wi'xName' 'Arial'
      'ex.rng.fnt'⎕wi'size'12
      'ex.rng.fnt'⎕wi'Bold'1

        'ex.wksht'⎕wi'Range>ex.rng' 'B1'  ⍝ changement de largeur de colonne
          'ex.rng'⎕wi'ColumnWidth'16

        'ex.wksht'⎕wi'xName' 'Ventes'    ⍝ changement du nom de la feuille

              'ex'⎕wi'caption'(AV2ANSI'Démonstration APL+WinObjet à AFAPL')

         'ex.wkbk'⎕wi'SaveAs' 'c:\temp\afapl.xls'  ⍝ sauvegarde du classeur
        'ex.wkbks'⎕wi'XOpen' 'c:\temp\temp.xls' ⍝ chargement d'un autre classeur
    ∇
```

Créer vos propres objets en APL+Win

Avec la nouvelle version 3.5, **APL+Win** va plus loin et vous permet de créer vos propres objets, au niveau système.

C'est à dire que vous allez pouvoir définir leurs **propriétés**, **méthodes** et **événements**, puis vous servir de la fonction ⎕*WI* pour utiliser vos objets comme s'ils s'agissaient d'objets Windows standard.

Ce qui est remarquable, c'est la **pureté** et la **simplicité** du système conçu par APL2000 pour vous permettre de créer vos objets.

Les concepts de base de ce nouveau système sont les suivants: il suffit d'ajouter à APL+Win deux nouveaux événements:

> Un événement *onNew* qui se produit chaque fois que l'on se sert de ⎕*WI'New'*. En interceptant la demande de création d'un objet, avant qu'il ne soit créé par le système, il est ainsi possible de le créer soi-même avec toutes les caractéristiques souhaitées, en s'appuyant sur un ou plusieurs objets existants dans APL+Win
>
> Un événement *onAction* qui se produit chaque fois que vous tentez d'utiliser une propriété ou une méthode sur votre propre objet, avant que le système APL+Win ne tente d'exécuter cette propriété ou méthode

Grâce à ces 2 idées de génie, APL2000 a pu implanter un système qui vous permette de créer vos propres objets, quelque soit leur nature et leur complexité, sans alourdir son interface Windows actuelle, sans modifier vos habitudes de programmation, sans ajouter la moindre fonction ou variable système à APL+Win.

Avec le système qu'ils ont créé, vous pouvez commencer à bâtir une hiérarchie d'objets aussi complète et sophistiquée que celle de **Delphi** par exemple.

Vous pourrez utiliser l'héritage, le polymorphisme, etc...

Vous pourrez enfin réellement réutiliser vos objets d'une application à l'autre, sans avoir à "réinventer la roue" à chaque fois.

Après avoir expliqué la création d'objets personnalisés en APL+Win, je vous montrerai quelques exemples d'objets extraits de ma propre bibliothèque d'objets.

Comment créer un objet personnalisé en APL+Win 3.5

La meilleure façon de démontrer est (comme toujours) de prendre un exemple.

Tous les langages de développement Windows possèdent un objet "**Edit**" ou "zone de texte" pour permettre la saisie à l'écran. Tous possèdent également un objet "**Label**" ou "zone de texte statique" pour permettre l'affichage de texte dans une fenêtre.
Lorsque vous voulez créer une zone de saisie dans une fenêtre, vous devez la plupart du temps associer une zone "**Edit**" et un "**Label**".

Exemple:

```
⎕wself←'ff.lab'⎕wi'Create' 'Label'('caption' 'Nom')('w
here'10 10 20 30)

⎕wself←'ff.edi'⎕wi'Create' 'Edit'('where'10 43 20 100)
```



Comme vous le constatez, il n'est pas très difficile de créer une fenêtre avec un **Label** et un **Edit**.

Malgré tout, il faut 2 instructions puisqu'il y a 2 objets à créer (en dehors de la fenêtre). Par ailleurs nous constatons que le texte du **Label** n'est pas à la même hauteur que le texte saisi. Enfin, si nous devions déplacer la zone de saisie, en mode "design" ou par programme, il faudrait effectuer un second déplacement similaire pour le **Label**, avec le risque de ne pas le repositionner tout à fait correctement par rapport à la zone **Edit**.

Egalement, si nous souhaitions changer la police des éléments de notre fenêtre, il faudrait effectuer deux opérations: un changement de police pour le **Label** et un changement de police pour la zone **Edit**.

Toutes ces remarques nous amènent à imaginer un nouveau type d'objet qui serait la combinaison d'un **Label** et d'un **Edit**, toujours parfaitement solidaires l'un de l'autre et parfaitement positionnés l'un par rapport à l'autre.

Tentons de créer un tel objet.

Choisissons **TLabelEdit** comme nom de classe pour ce nouveau type d'objet.

La première chose à faire est de déclarer ce nouvel objet de façon à ce qu'il soit connu d'APL+Win.

Ceci est réalisé en ajoutant ce nom de classe d'objet à la propriété **newclasses** de l'objet système d'APL+Win.

```
'#'⎕wi'newclasses' 'TLabelEdit'
```

Maintenant définissons une gestion d'événement pour l'événement **OnNew**:

```
'#'⎕wi'onNew' 'TLabelEdit"New"'
```

Ceci signifie que lorsque nous tenterons de créer une instance de notre objet **TLabelEdit** en écrivant:

```
'ff.le'⎕wi'New' 'TLabelEdit'
```

l'événement **onNew** se déclenchera et exécutera la fonction **TLabelEdit** avec un argument droit égal à **'New'.**

Vous remarquerez que nous prenons grand soin de nommer la fonction qui gèrera notre nouvel objet, du même nom exact que la classe de cet objet. En effet, lorsque nous aurons créé des dizaines ou centaines d'objets personnalisés, cela facilitera la maintenance. Par ailleurs, en **APL+Win**, lorsque l'on appuie sur **Ctrl+Shift+O** lorsque le curseur est sur un nom de fonction, cela ouvre la fonction dans l'éditeur: il sera donc très pratique de simplement mettre le curseur sur le mot **TLabelEdit** à l'écran et de faire **Ctrl+Shift+O** pour éditer la fonction qui gère cet objet.

Création d'un nouvel objet

Nous allons donc développer une fonction **TLabelEdit** correspondant à notre objet, et nous allons nous efforcer de placer la totalité de notre objet dans cette fonction (propriétés, méthodes, gestion d'événements, ...) sans appeler de sous-programmes.

De cette façon notre objet sera facilement réutilisable. Il suffira de copier sa fonction dans une autre application.

Le squelette que je recommande pour les fonctions d'objets personnalisés est le suivant:

```
    ∇ A TClass B;⎕io;⎕wself
[1]    ⍝∇ A TClass B -- TClass Template Object
[2]    ⍝∇ A ←→ object name
[3]    ⍝∇ B ←→ 'property'
[4]    ⍝∇    or 'property' value
[5]    ⍝∇    or 'Method'
[6]    ⍝∇    or 'Method' argument1 ... argumentN
[7]
[8]    ⎕io←1                                          ⍝ environnement
[9]    :if 2≠⎕nc'A' ⋄ A←⎕wself ⋄ :end                 ⍝ objet par défaut
[10]   :select B
[11]   :case'New'                                     ⍝ constructeur
[12]       ⎕wself←A ⎕wi'*Create' 'Class'              ⍝ création objet
[13]           ⎕wi'*onAction' 'TClass"Action"'        ⍝ onAction handler
[14]   :case'Action'
[15]       :select↑⎕warg
[16]       :case'class'
[17]           ⎕wres←'TClass'
[18]       :case'methods'
[19]           ⎕wres←(⎕wi'*methods')
[20]       :case'properties'
[21]           ⎕wres←(⎕wi'*properties')
[22]       :else
[23]           ⎕wres←⎕wi'*'
[24]       :end
[25]   :else
[26]       ⎕error'Unkown TClass command: ',B
[27]   :end
     ∇
```

L'argument droit de nos fonctions d'objets seront des noms de **propriétés** (suivies de valeurs ou non), des noms de **méthodes** (suivies d'arguments ou non), des noms **d'événements**.

L'argument gauche de nos fonctions d'objet sera toujours le nom de l'**instance** d'objet que nous souhaitons créer.

En ligne 9, s'il n'y avait pas d'argument gauche passé à la fonction d'objet, nous récupérerions le nom de l'objet dans la variable système ⎕WSELF. Ensuite, le reste de la fonction est une clause **:select** dont les différents **:case** comprendront toujours:

Le **:case'New'** ou "constructeur" qui contiendra les lignes de code servant à créer une instance de notre objet
Le **:case'Action'** qui recevra et gèrera tous les appels à des propriétés ou méthodes de l'objet

Le **:else** qui renverra une erreur en cas de non reconnaissance du nom de la propriété ou méthode invoquée.

Que mettre donc dans le constructeur de notre objet? Eh bien, il nous faut créer deux objets, un **Label** et un **Edit**.

Mais nous souhaitons nous assurer qu'ils possèdent tous les deux une échelle **"pixels"**, ce qui se fait en fixant leurs propriété **'scale'** à **5**. Voici la fonction **TLabelEdit**, obtenue en remplaçant **TClass** par **TLabelEdit** partout dans la fonction **TClass**, puis en ajoutant les lignes **12** et **13**:

```
    ∇ A TLabelEdit B;⎕io;⎕wself
[1]    ⍝∇ A TLabelEdit B -- TLabelEdit Template Object
[2]    ⍝∇ A ←→ object name
[3]    ⍝∇ B ←→ 'property'
[4]    ⍝∇   or 'property' value
[5]    ⍝∇   or 'Method'
[6]    ⍝∇   or 'Method' argument1 ... argumentN
[7]
[8]    ⎕io←1                                       ⍝ environment
[9]    :if 2≠⎕nc'A' ⋄ A←⎕wself ⋄ :end              ⍝ default object
[10]   :select B
[11]   :case'New'                                  ⍝ constructor
[12]       ⎕wself←A ⎕wi'*Create' 'Edit'('scale'5) ⍝ create an Edit object
[13]       ⎕wself←(A,'Lab')⎕wi'*Create' 'Label'('scale'5) ⍝ now a Label
[14]           ⎕wi'*onAction' 'TLabelEdit"Action"' ⍝ onAction handler
[15]   :case'Action'
[16]       :select↑⎕warg
[17]       :case'class'
[18]           ⎕wres←'TLabelEdit'
[19]       :case'methods'
[20]           ⎕wres←(⎕wi'*methods')
[21]       :case'properties'
[22]           ⎕wres←(⎕wi'*properties')
[23]       :else
[24]           ⎕wres←⎕wi'*'
[25]       :end
[26]   :else
[27]       ⎕error'Unkown TLabelEdit command: ',B
[28]   :end
    ∇
```

Nous pouvons désormais créer une instance de notre objet:

```
'ff.le'⎕wi'New' 'TLabelEdit'
```

Toutefois les 2 sous-objets qui le composent se trouvent positionnés au centre de l'écran et sont superposés.

Nous souhaitons que notre objet **TLabelEdit** dispose d'une propriété **where** qui nous permettent de les positionner correctement sur notre fenêtre.

Création d'une propriété

Il nous faut d'abord pouvoir préciser quel est le texte du **Label** associé au contrôle **Edit**. Il faut pour cela que nous puissions changer la propriété **caption** du **Label**. Mais n'oublions pas que notre objet est un ensemble **Label + Edit** et qu'il ne connait pas lui-même de propriété **caption**.

Il faut donc en définir une.

La définition d'une propriété d'un objet personnel est très simple. Il suffit d'ajouter un **:case'nom_de_propriété'** au **:select** du **:case'Action'** de l'objet.

Toutefois pour une propriété il faut traiter 2 cas:

L'interrogation de la valeur actuelle de la propriété
Le changement de valeur de la propriété

Ceci sera très simplement réalisé à l'aide d'une structure **:if ... :else ... :end**, comme suit:

```
:case'caption'
    :if 1=⍴⎕warg
        ⎕wres←(A,'Lab')⎕wi'caption'
    :else
        (A,'Lab')⎕wi'caption'(2⊃⎕warg)
    :end
```

Vous noterez 2 points essentiels.

Lors de l'utilisation d'une propriété ou d'une méthode, la variable système `⎕warg` contient toujours en premier élément le nom de la propriété ou méthode invoquée, et dans les éléments suivants, la nouvelle valeur de la propriété ou les arguments de la méthode.

Par exemple, si nous exécutions:

```
'ff.le'⎕wi'caption' 'Essai'
```

`⎕warg` contiendrait un vecteur généralisé de 2 chaines de caractères, comme le montre l'instruction suivante:

```
      ]display ⎕warg
.→---------------.
| .→------. .→----. |
| |caption| |Essai| |
| '-------' '-----' |
'∊---------------'
```

Par ailleurs, il faut placer dans la variable système `⎕wres` ce que nous désirons recevoir comme résultat.

La clause **:if** correspond à l'interrogation de la valeur actuelle de la propriété **caption** et la clause **:else** à sa modification.

Toutes les propriétés d'un objet doivent être implémentées de cette façon.

Après avoir ajouté le **:case'caption'** à notre objet **TLabelEdit**, nous pouvons utiliser la propriété **caption** de notre objet:

```
      'ff.le'⎕wi'caption'
leLab
```

En **APL+Win**, la **caption** par défaut d'un **Label** est égale à son nom.

```
      'ff.le'⎕wi'caption' 'Nom du salarié'
```

Le changement de valeur d'une propriété ne rend pas de résultat.

Comme vous le constatez, l'implémentation d'une propriété d'un objet est une tâche relativement simple.

Création d'une méthode

Pour calculer la position exacte du contrôle **Edit**, il nous faudra connaître la longueur exacte en pixels du **Label**.

Pour ce faire nous pouvons écrire une fonction utilitaire **LabelSize** dont voici le code:

```
    ∇ R←LabelSize L;F;C;D;E;F;⎕io
[1]  ⍝∇ R←LabelSize L- Calcule les dimensions en pixels du Label
[2]  ⍝∇ L ↔ nom de l'objet Label
[3]  ⍝∇ R ↔ longueur du label en pixels
[4]
[5]  ⎕io←1
[6]  C←L ⎕wi'caption'                ⍝ texte du Label
[7]  D←(¯1+L⍳'.')↑L                  ⍝ nom de la fenêtre
[8]  E←D ⎕wi'scale'                  ⍝ échelle de la fenêtre
[9]  D ⎕wi'scale'5                   ⍝ force échelle "pixels"
[10] :if 0∊⍴F←L ⎕wi'font'            ⍝ si pas de police définie
[11]     F←'MS Sans Serif'8 0        ⍝ police par défaut
[12] :end                            ⍝ fin
[13] R←∊D ⎕wi'Draw'((⊂'Font'),F)((⊂'?Text'),C) ⍝ longueur Label
[14] D ⎕wi'scale'E                   ⍝ rétablie échelle fenêtre
    ∇
```

Exemple:

```
      LabelSize'ff.leLab'
13 22
```

Toutefois, de façon à éviter tout sous programme pour notre objet **TLabelEdit**, il est souhaitable d'implémenter cette fonction utilitaire en tant que **méthode** de l'objet.

Comme pour l'ajout d'une propriété, l'ajout d'une méthode à notre objet consiste à ajouter un **:case'nom_de_méthode'** au **:select** du **:case'Action'** de l'objet.

Ajoutons donc le code de la fonction **LabelSize** à notre objet **TLabelEdit**. Notre fonction **TLabelEdit** devient:

```
 ∇ A TLabelEdit B;C;D;E;F;L;⎕io;⎕wself
    ⍝∇ A TLabelEdit B -- TLabelEdit Template Object
    ⍝∇ A ←→ object name
    ⍝∇ B ←→ 'property'
    ⍝∇   or 'property' value
    ⍝∇   or 'Method'
    ⍝∇   or 'Method' argument1 ... argumentN

    ⎕io←1                               ⍝ environnement
    :if 2≠⎕nc'A' ⋄ A←⎕wself ⋄ :end      ⍝ objet par défaut
    :select B
    :case'New'                          ⍝ constructeurr
        ⎕wself←A ⎕wi'*Create' 'Edit'('scale'5)
        ⎕wself←(A,'Lab')⎕wi'*Create' 'Label'('scale'5)
        A ⎕wi'*onAction' 'TLabelEdit"Action"'
    :case'Action'
        :select↑⎕warg
        :case'class'
            ⎕wres←'TLabelEdit'
        :case'methods'
            ⎕wres←(⎕wi'*methods')
        :case'properties'
            ⎕wres←(⎕wi'*properties')
        :case'caption'                  ⍝ propriété caption
            :if 1=⍴⎕warg
                ⎕wres←(A,'Lab')⎕wi'caption'
            :else
                (A,'Lab')⎕wi'caption'(2⊃⎕warg)
            :end
        :case'LabelSize'                ⍝ méthode LabelSize
            L←A,'Lab'                   ⍝ nom du Label
            C←L ⎕wi'caption'            ⍝ texte du Label
            D←(¯1+L⍳'.')↑L              ⍝ nom de la fenêtre
            E←D ⎕wi'scale'              ⍝ échelle de la fenêtre
            D ⎕wi'scale'5               ⍝ force échelle pixels
            :if 0∊⍴F←L ⎕wi'font'        ⍝ si police non définie
                F←'MS Sans Serif'8 0    ⍝ police par défaut
            :end                        ⍝ fin
            ⎕wres←∊D ⎕wi'Draw'((⊂'Font'),F)((⊂'?Text'),C)
            D ⎕wi'scale'E               ⍝ rétablit échelle fen.
        :else
            ⎕wres←⎕wi'*'
        :end
    :else
        ⎕error'Unkown TLabelEdit command: ',B
    :end
   ∇
```

Utilisons notre nouvelle méthode:

```
      'ff.le'⎕wi'LabelSize'
13 69
```

Tout marche bien: notre **Label** a une longueur de **69** pixels et une hauteur de **13** pixels.

Il nous faut maintenant implémenter une propriété **where** pour pouvoir positionner parfaitement le **Label** et son Contrôle **Edit** associé.

Création de la propriété where

Créons donc la propriété **where** pour notre objet.

Lorsque nous définirons une position dans l'écran pour **TLabelEdit**, ce sera celle du coin supérieur gauche du **Label**. La position du **Edit** devra se déduire automatiquement de celle du **Label** en respectant:

Le fait que le texte du **Label** devra être à la même hauteur que le texte du contrôle **Edit**

Le fait que le contrôle **Edit** devra commencer 5 pixels après le dernier caractère du **Label**

Voici comment implanter la propriété **where**:

```
[47]     :case'where'                         ⍝ propriété where
[48]         L←A,'Lab'                        ⍝ nom du Label
[49]         D←C←∊1↑⎕warg                     ⍝ argument de where
[50]         :if 4≠⍴C                         ⍝ contrôle argument
[51]             ⎕wres←'La propriété where doit contenir 4 éléments'
[52]             :return                      ⍝ sortie si erreur
[53]         :end
[54]         (E F)←⎕wi'LabelSize'L            ⍝ dimensions du label
[55]         D[1]←C[1]-3                      ⍝ position vert. du Edit
[56]         D[2]←C[2]+F+5                    ⍝ position horz. du Edit
[57]         D[3]←E+7                         ⍝ ajuste hauteur du Edit
[58]         D[4]←C[4]                        ⍝ longueur du Edit
[59]         C[3 4]←E F                       ⍝ dimensions du Label
[60]         L ⎕wi'*where'C                   ⍝ positionne le Label
[61]         A ⎕wi'*where'D                   ⍝ positionne le Edit
```

Le principe consiste à récupérer la valeur de la propriété **where** de notre objet **TLabelEdit** (ligne **49**), à affecter cette propriété au **Label** (ligne **60**), à récupérer la longueur exacte en pixels de notre **Label** (ligne **54** qui appelle notre méthode **LabelSize**), puis à ajuster précisément la propriété **where** du contrôle **Edit** associé au **Label**, en fonction de la propriété **where du Label** (lignes **55** à **58**) et enfin de forcer les dimensions du **Label** à être précisément celles de sa **caption** (ligne **59**).

Vous noterez la présence des étoiles devant la propriété **where** en lignes **42** et **43**.

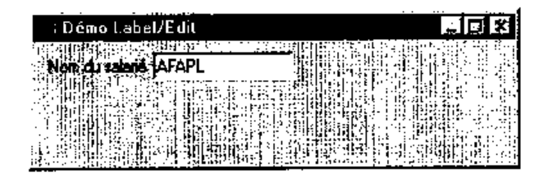
Ceci est en effet nécessaire pour la raison suivante: nous avons implémenté une propriété **where** pour notre objet **TLabelEdit** qui porte le même nom qu'une propriété existante dans APL+Win (**where**) pour les contrôles **Label** et **Edit**.

La présence d'une étoile devant **where** en **ligne 42 et 43** indique à APL+Win d'exécuter la propriété **where** standard des **Label** et **Edit** et non d'appeler à nouveau la propriété **where** que nous venons de définir dans notre objet **TLabelEdit**.

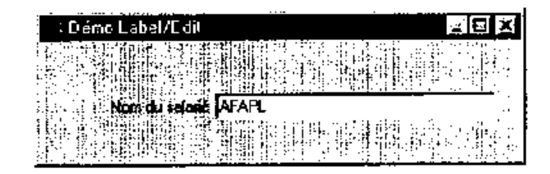
Si nous n'avions pas mis d'étoile (*) devant **where** en **ligne 43**, notre programme entrerait dans une boucle sans fin!

Essayons notre propriété **where**:

```
'ff.le'⎕wi'where'10 10 15 100
```




```
'ff.le'⎕wi'where'40 50 15 200
```



Vous noterez que la zone **Edit** est parfaitement positionnée dans les 2 cas par rapport au **Label**.

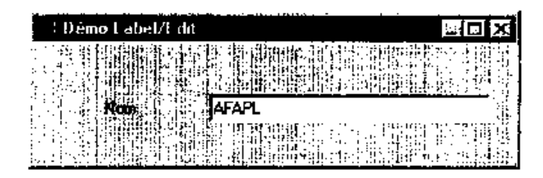
Le texte de la zone **Edit** est exactement à la même hauteur que le texte du **Label**.

La zone **Edit** commence exactement à la distance souhaitée après le **Label**.

Enfin, le dernier élément de notre propriété **where** fixe la longueur du contrôle **Edit**. Il était en effet inutile de l'appliquer à la longueur du **Label** puisque celle-ci est calculée et modifiée dans le code de la propriété **where** de **TLabelEdit**.

Maintenant, que se passe-t-il si nous modifions la propriété **caption** de notre **Label**:

```
'ff.le'⎕wi'caption' 'Nom'
```



Comme vous le constatez, la position de la zone **Edit** n'est pas ajustée automatiquement pour être placée juste après le **Label**.

Comment pouvons-nous remédier à ce petit problème de notre objet **TLabelEdit**.

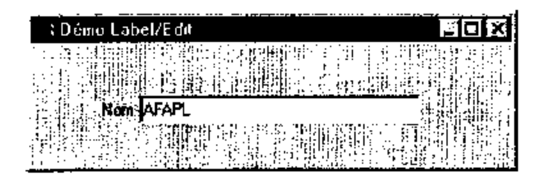
Il suffit d'ajouter l'instruction suivante à la propriété **caption** de notre objet **TLabelEdit**:

```
A ⎕wi'where'((3↑(A,'Lab')⎕wi'*where'),4⊃A ⎕wi'*where')
```

En effet, le fait de changer la propriété **where** de notre objet **TLabelEdit** permet de recalculer parfaitement la position du **Label** et du **Edit**. Comme nous ne voulons pas changer la position du **Label**, nous lui réaffectons la position actuelle (soit: `(3↑(A,'Lab')⎕wi'*where')`). Il nous faut seulement récupérer la largeur de notre contrôle **Edit** (soit: `4⊃A ⎕wi'*where'`). Vous noterez à nouveau l'utilisation des étoiles pour indiquer que nous voulons utiliser la propriété **where** standard d'**APL+Win** et non la propriété **where** de notre objet **TLabelEdit**.

Réessayons maintenant de modifier la **caption** de notre objet:

```
'ff.le'⎕wi'caption' 'Nom'
```



Notre objet **Edit** se repositionne maintenant parfaitement, juste après le **Label** en tenant compte de la nouvelle longueur de ce dernier.

A vrai dire, nous commençons à disposer d'un nouvel objet personnalisé (**TLabelEdit**) qui correspond exactement à nos désirs. Il nous permettra d'aller beaucoup plus vite pour définir des fenêtres de saisie et de plus nous permettra d'obtenir une interface parfaite au pixel près.

Bien d'autres propriétés et méthodes pourraient être ajoutées à cet objet, mais je souhaitais simplement vous montrer les principes du développement d'objet personnalisés en APL+Win 3.5.

Ajout du Destructeur

Il nous reste toutefois une fonctionnalité indispensable à implémenter dans notre objet **TLabelEdit**: la destruction de l'objet.

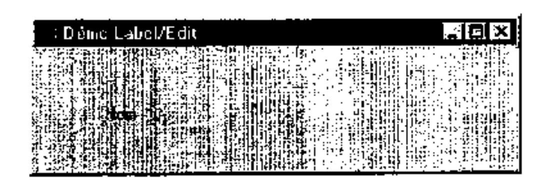
En effet, de même que tout objet personnalisé doit comporter un **constructeur** (méthode **'New'**), tout objet doit en principe comporter un **destructeur**.

Si nous n'ajoutions pas de destructeur à notre objet, nous n'effectuerions pas une destruction propre de l'objet en écrivant:

```
'ff.le'⎕wi'*Delete'
```

En effet cette instruction détruit le contrôle **Edit** (qui porte le même nom que notre objet) mais pas le **Label**, qui s'appelle **ff.leLab**.

La preuve en est le résultat de cette destruction:



Pour résoudre ce problème il nous faut gérer l'événement **onDelete** sur l'objet **Edit** et, dans la fonction de gestion de cet événement, il nous faut détruire l'objet **Label**.

Voici le code nécessaire:

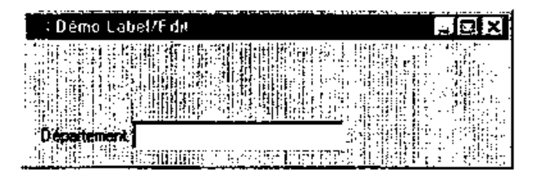
```
:case'New'                                    ⍝ constructeur
    ⎕wself←A ⎕wi'*Create' 'Edit'('scale'5)
         ⎕wi'onDelete' "TLabelEdit'onDelete'"
    ⎕wself←(A,'Lab')⎕wi'*Create' 'Label'('scale'5)
    A ⎕wi'*onAction' 'TLabelEdit"Action"'
:case'onDelete'                               ⍝ destructeur
    (A,'Lab')⎕wi'*Delete'
```

Ainsi, la destruction de l'objet **TLabelEdit** est parfaite.

Cette même technique de destruction, utilisant l'événement **onDelete** du sous-objet principal (ici le contrôle **Edit**) d'un objet personnalisé doit être appliquée systématiquement pour détruire les sous-objets associés (ici le **Label**).

Essayons notre nouveau **destructeur**. Mais d'abord recréons, en une seule instruction, un objet **TLabelEdit** dans notre fenêtre:

```
'ff.le'⎕wi'*Create' 'TLabelEdit'('caption' 'Département
')('where'60 10 15 150)
```



Et maintenant détruisons le:

```
'ff.le'⎕wi'*Delete'
```



Cette fois-ci la destruction est parfaite.

Création d'un événement

Supposons que dans une application utilisant l'objet **TLabelEdit**, nous désirions pouvoir réagir à toute modification de la propriété **police** de notre objet **TLabelEdit**.

Tout d'abord, il nous faut créer une propriété police pour notre objet **TLabelEdit**, qui modifie à la fois la police du **Label** et la police du **Edit**.

En voici le code:

```
[36] :case'font'
[37]      :if 1=⍴⎕warg
[38]           ⎕wres←(A,'Lab')⎕wi'*font'
[39]      :else
[40]           C←1↓⎕warg
[41]           L←A,'Lab'
[42]           L ⎕wi'*font'C
[43]           A ⎕wi'*font'C
[44]           A ⎕wi'where'((3↑L ⎕wi'*where'),4⊃A ⎕wi'*where')
[45]           A ⎕wi'*Event' "TLabelEdit'onFontChange'" 'onFontChan
ge'(1↓⎕warg)
[46]      :end
```

En ligne **45** nous avons défini un nouvel événement qui se produira donc à chaque fois que l'utilisateur ou l'application modifiera la propriété **font** de l'objet **TLabelEdit**.

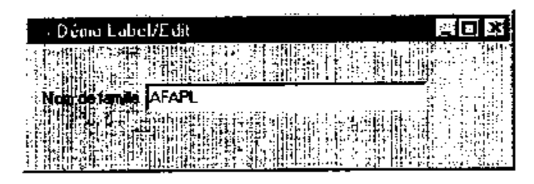
Cette instruction utilise la nouvelle méthode **Event** (disponible pour tout objet d'**APL+Win 3.5**) pour indiquer au système de générer immédiatement un événement **onFontChange** qui sera géré par l'expression **TLabelEdit'onFontChange'**.

Ajoutons donc un **:case'onFontChange'** à notre objet pour gérer cet événement.

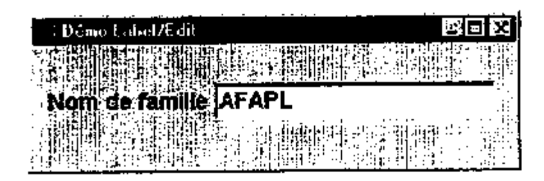
```
:case'onFontChange'                    ⍝ événement personnalisé
    ⎕←"La police de l'objet ",⎕wself," a changé et est devenue ",
⎕warg
```

Essayons maintenant de modifier la police de notre objet:

```
'ff.le'⎕wi'*Create' 'TLabelEdit'('caption' 'Nom de famille')
('where'30 10 10 200)
```



```
'ff.le'⎕wi'font' 'Arial'20 1
La police de l'objet ff.le a changé et est devenue  Arial 20 1
```



Vous remarquerez par ailleurs que la propriété **where** de notre objet a été parfaitement conçue puisqu'elle a permis de réadapter automatiquement la hauteur du **Label** et du **Edit**, ainsi que leurs positions relatives, sans que nous n'ayions eu a ajouter le moindre code pour ce faire.

Le texte du **Label** et du **Edit** restent à la même hauteur, au pixel près.

Nous nous somme contentés d'afficher dans la session **APL** une information sur l'événement de changement de police, mais bien sûr, nous aurions pu développer tout autre code **APL** pour gérer l'événement de changement de police.

Nous pourrions aller plus loin. Par exemple, lorsque nous changeons la police de la fenêtre, ce qui change par héritage la police des contrôles de la fenêtre, donc du **Label** et du contrôle **Edit**, nous pourrions définir un évènement **onFontChange** sur la fenêtre qui permette de redimensionner le **Label** et le contrôle **Edit**.

Je vous laisse le soin de mettre en place cette fonctionnalité, à titre d'exercice.
La page suivante nous reproduit le code complet de la fonction **TlabelEdit**.

Code de l'objet TLabelEdit

```
    ∇ A TLabelEdit B;C;D;E;F;L;⎕io;⎕wself;G;H
  ⍝∇ A TLabelEdit B -- TLabelEdit Template Object
  ⍝∇ A ←→ object name
  ⍝∇ B ←→ 'property'
  ⍝∇    or 'property' value
  ⍝∇    or 'Method'
  ⍝∇    or 'Method' argument1 ... argumentN

  ⎕io←1                                  ⍝ environnement
  :if 2≠⎕nc'A' ⋄ A←⎕wself ⋄ :end         ⍝ objet par défaut
  :select B
  :case'New'                             ⍝ constructeur
      ⎕wself←A ⎕wi'*Create' 'Edit'('scale'5)
          ⎕wi'onDelete' "TLabelEdit'onDelete'"
      ⎕wself←(A,'Lab')⎕wi'*Create' 'Label'('scale'5)
      A ⎕wi'*onAction' 'TLabelEdit"Action"'
  :case'onFontChange'                    ⍝ événement personnalisé
      ⎕←"La police de l'objet ",⎕wself," a changé et est devenue ",⎕warg
  :case'onDelete'                        ⍝ destructeur
      (A,'Lab')⎕wi'*Delete'
  :case'Action'
      :select↑⎕warg
      :case'class'
          ⎕wres←'TLabelEdit'
      :case'methods'
          ⎕wres←(⎕wi'*methods')
      :case'properties'
          ⎕wres←(⎕wi'*properties')
      :case'caption'                     ⍝ propriété caption
          :if 1=⍴⎕warg
              ⎕wres←(A,'Lab')⎕wi'caption'
          :else
              L←A,'Lab'
              L ⎕wi'caption'(2⊃⎕warg)
              A ⎕wi'where'((3↑L ⎕wi'*where'),4⊃A ⎕wi'*where')
          :end
      :case'font'
          :if 1=⍴⎕warg
              ⎕wres←(A,'Lab')⎕wi'*font'
          :else
              C←1↓⎕warg
              L←A,'Lab'
              L ⎕wi'*font'C
              A ⎕wi'*font'C
              A ⎕wi'where'((3↑L ⎕wi'*where'),4⊃A ⎕wi'*where')
              A ⎕wi'*Event' "TLabelEdit'onFontChange'" ''(1↓⎕warg)
          :end
      :case'where'                       ⍝ propriété where
          L←A,'Lab'                      ⍝ nom du Label
          D←C←∊1↓⎕warg                   ⍝ argument de where
          :if 4≠⍴C                       ⍝ contrôle argument
              ⎕wres←'La propriété where doit contenir 4 éléments'
               :return                   ⍝ sortie si erreur
          :end
```

```
            (E F)←⎕wi'LabelSize'L        ⍝ dimensions du label
            D[1]←C[1]-3                  ⍝ position vert. du Edit
            D[2]←C[2]+F+5                ⍝ position horz. du Edit
            D[3]←E+7                     ⍝ ajuste hauteur du Edit
            D[4]←C[4]                    ⍝ longueur du Edit
            C[3 4]←E F                   ⍝ dimensions du Label
            L ⎕wi'*where'C               ⍝ positionne le Label
            A ⎕wi'*where'D               ⍝ positionne le Edit
        :case'Destroy'                   ⍝ méthode Destroy
            (A,'Lab')⎕wi'*Delete'        ⍝ destruction du Label
            A ⎕wi'*Delete'               ⍝ destruction du Edit
        :case'LabelSize'                 ⍝ méthode LabelSize
            L←A,'Lab'                    ⍝ nom du Label
            C←L ⎕wi'caption'             ⍝ texte du Label
            D←(¯1+L⍳'.')↑L               ⍝ nom de la fenêtre
            E←D ⎕wi'scale'               ⍝ échelle de la fenêtre
            D ⎕wi'scale'5                ⍝ force échelle "pixels"
            :if 0∊⍴F←L ⎕wi'font'         ⍝ si police non définie
                F←'MS Sans Serif'8 0     ⍝ police par défaut
            :end                         ⍝ fin
            ⎕wres←∊D ⎕wi'Draw'((⊂'Font'),F)((⊂'?Text'),C)
            D ⎕wi'scale'E                ⍝ rétablit échelle fenê.
        :else
            ⎕wres←⎕wi'*'
        :end
    :else
        ⎕error'Unkown TLabelEdit command: ',B
    :end
  ∇
```

Développement d'objets non visuels

Il est parfaitement possible de créer des objets non visuels en APL+Win 3.5.

Par exemple un objet "**Base de Données Relationnelle**" ou un objet "**Chronomètre**" par exemple.

Sans entrer dans les détails, sachez qu'il suffit de s'appuyer sur un objet visuel tout en laissant cet objet visuel caché.

L'objet visuel qui demande le moins de ressources système lors de sa création est un objet **Menu popup**.

La création d'un tel objet est très simple:

```
'mnu'⎕wi'New' 'Menu'
```

d'où le modèle général d'objet non visuel en **APL+Win 3.5**:

```
    ∇ A TNonVisualClass B;C;⎕io;⎕wself
      ⍝∇ A TNonVisualClass B -- TNonVisualClass Template Object
      ⍝∇ A ↔ object name
      ⍝∇ B ↔ 'property'
      ⍝∇   or 'property' value
      ⍝∇   or 'Method'
      ⍝∇   or 'Method' argument1 ... argumentN

      ⎕io←1                                   ⍝ environnement
      :if 2≠⎕nc'A' ⋄ A←⎕wself ⋄ :end          ⍝ objet par défaut
      :select B
      :case'New'                              ⍝ constructeur
          ⎕wself←A ⎕wi'*Create' 'Menu'        ⍝ création objet non visuel
             ⎕wi'*onAction' 'TNonVisualClass"Action"' ⍝ onAction handler
      :case'Action'
          :select↑⎕warg
          :case'class'
              ⎕wres←'TNonVisualClass'
          :case'methods'
              ⎕wres←(⎕wi'*methods')
          :case'properties'
              ⎕wres←(⎕wi'*properties'),'attach' 'wherelc'
          :else
              ⎕wres←⎕wi'*'
          :end
      :else
          ⎕error'Unkown TNonVisualClass command: ',B
      :end
    ∇
```

Héritage

La possibilité de créer ses propres objets devient vraiment passionnante lorsque l'on peut faire intervenir la notion **d'héritage**.

Je réserve le développement détaillé de cette notion d'héritage à un prochain article. En attendant, les passionnés pourront réfléchir à la façon d'implémenter **l'héritage** dans les modèles d'objets décrits précédemment.

Le principe de **l'héritage** est le suivant.

Nous avons défini un objet générique **TLabelEdit** avec quelques **propriétés** et **méthodes**.

Supposons dans une application que nous ayons besoin d'un objet disposant de toutes les fonctionnalités de **TLabelEdit**, mais qui de plus n'accepte que de la **saisie numérique**.

Deux solutions s'offrent à nous:

Soit copier la fonction **TLabelEdit** sous le nom **TLabelEditNum** par exemple, puis lui rajouter la fonctionnalité nécessaire
Soit créer un nouvel objet de taille minimum, qui hérite de l'objet **TLabelEdit** et de toutes ses fonctionnalités, mais qui en plus sache rejeter toute saisie non numérique

La première solution est à rejeter définitivement et catégoriquement: c'est celle que nous avons tous employé pendant des années, jusqu'à l'avènement de la programmation Objet.

Pourquoi la rejeter?

C'est très simple: si vous êtes amené à modifier la fonction **TLabelEdit**, vous devrez répéter les mêmes modifications dans son clone **TLabelEditNum**. Et si vous avez créé 10 fonctions clones de **TLabelEdit**, vous devrez répéter les modifications 10 fois. Si vous ne le faites pas, vos différentes versions de **TLabelEdit** vont commencer à diverger avec tous les problèmes de maintenance que cela entraine:

Quelle est la bonne version?
Comment réunifier toutes les versions de la même fonction?
Etc.

La bonne solution est donc d'utiliser **l'héritage**.

L'héritage est une fonctionnalité, lorsqu'on la maitrise, qui est absolument extraordinaire, presque magique.

Lorsque vous savez faire hériter un objet d'un autre objet, vous pouvez alors songer, enfin, à construire une hiérarchie complète d'objets, descendant d'un objet générique, par exemple appelé **TObject**. Vous pouvez ainsi construire un arbre d'objets tous les objets héritant automatiquement des fonctionnalités de leurs parents, grand-parents, etc.

Le développement d'un tel ensemble d'objet est certes un très gros travail, mais les bénéfices pour le développeur sont immenses:

gains de temps de développements phénoménaux
sécurité des applications considérablement accrue
taille du code de l'application sensiblement réduite
facilité de développement grandement améliorée

Quelques exemples d'objet

Pour terminer permettez moi de vous présenter quelques exemples d'objets développés en APL+Win 3.5, extraits de ma propre librairie d'objets APL.

Par manque de place, je ne ferais pas ou peu de commentaires sur ces objets, mais vous montrerai simplement leur utilisation.

L'objet TAgent

```
'ag'⎕wi'*Create' 'TAgent'
'ag'⎕wi'Merlin' 'New'
'ag'⎕wi'Merlin' 'Show'
```



Merlin est un personnage animé, capable de très nombreux comportements, capable de parler dans plusieurs langues, de comprendre et de réagir à la parole, d'afficher ce qu'il prononce dans des bulles, de jouer de la musique, etc... Voici un extrait de ses comportements

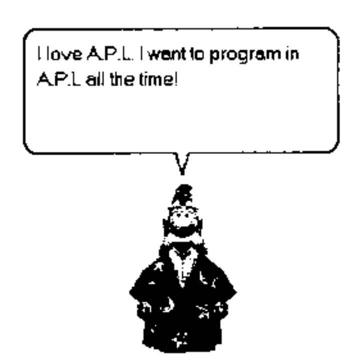
```
      'ag'⎕wi'Merlin' 'PlayMethods'
Acknowledge Alert Announce Blink Confused Congratulate Congratulate_2
      Decline DoMagic1 DoMagic2 DontRecognize Explain GestureDown Ges
      tureLeft GestureRight GestureUp GetAttention* GetAttention*Conti
      nued Greet Hearing_1 Hearing_2 Hearing_3 Hearing_4 Hide Idle1_1
      Idle1_2 Idle1_3 Idle1_4 Idle2_1 Idle2_2 Idle3_1 Idle3_2 LookDown
      * LookDown*Blink LookLeft* LookLeft*Blink LookRight* LookRight*B
      link LookUp* LookUp*Blink MoveDown MoveLeft MoveRight MoveUp Ple
      ased Process Processing Read* Read*Continued Reading RestPose Sa
      d Search Searching Show StartListening StopListening Suggest Sur
      prised Think Thinking Uncertain Wave Write* Write*Continued Writ
      ing
```

Essayons quelques uns de ces comportements:

```
'ag'⎕wi'Merlin' 'Play' 'Announce'
```



```
      'ag'⎕wi'Merlin' 'Speak' 'I love A.P.L. I want to progr
am in A.P.L all the time!'
```

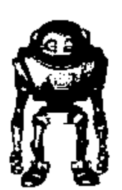


Merlin a 3 frères et soeurs. Faisons les apparaitre:

```
'ag'⎕wi'Peedy' 'New' ⋄ 'ag'⎕wi'Peedy' 'Show'
'ag'⎕wi'Robby' 'New' ⋄ 'ag'⎕wi'Robby' 'Show'
'ag'⎕wi'Genie' 'New' ⋄ 'ag'⎕wi'Genie' 'Show'

'ag'⎕wi'Peedy' 'MoveTo' 600 300

'ag'⎕wi'Peedy' 'Play' 'Idle3_3'
```

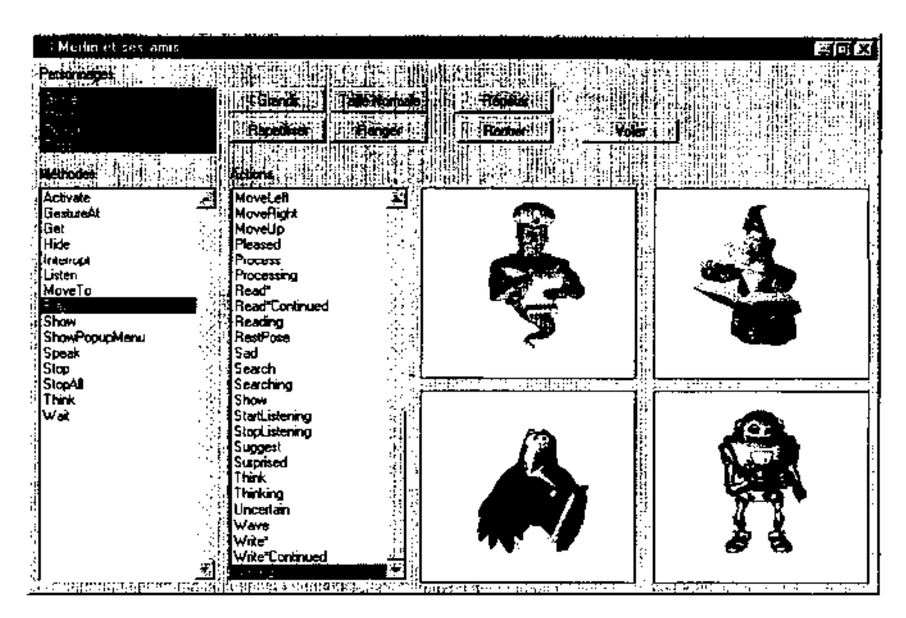






Pour ma fille de 8 ans, interessée par APL, langage qui permet d'animer de si drôles créatures, j'ai réalisé une petite interface en APL+Win pour lui permettre de s'amuser un peu. La voici:

```
      )load 54 merlin
54 MERLIN SAVED 04/10/1999 22:53:09
```



La fenêtre montre les 4 personnages en train d'écrire (exécution de la méthode **Writing**).

Maintenant voyons si ces personnages peuvent aider une petite fille de 8 ans à apprendre APL:

```
'ag'⎕wi'APLTutor'
```

Je me suis servi de ces personnages dans des applications pour certains de mes clients. Ils sont utilisés pour informer les utilisateurs des nouveautés de l'application au chargement de celle-ci.

L'objet TODBC

Tout aussi sérieux, l'objet **TODBC** est implanté entièrement en APL+Win et permet de travailler avec toute base de données relationnelle.

Il utilise les **API ODBC 3** de Windows. C'est un objet non visuel complexe, de **1722** lignes, mais à lui seul il donne accès à toutes les base de données relationnelles depuis APL+Win 3.5.

```
      'odbc'⎕wi'*Create' 'TODBC'

      'odbc'⎕wi'Open' 'TBM Access Database'
80084012 80084512

      'odbc'⎕wi'Tables'
 Qualifier            Owner Name                         Type
Remarks
 D:\tbm\Database\TBM        MSysACEs                     SYSTEM TABLE
 D:\tbm\Database\TBM        MSysColumns                  SYSTEM TABLE
 D:\tbm\Database\TBM        MSysIMEXColumns              SYSTEM TABLE
 D:\tbm\Database\TBM        MSysIMEXSpecs                SYSTEM TABLE
 D:\tbm\Database\TBM        MSysIndexes                  SYSTEM TABLE
 D:\tbm\Database\TBM        MSysMacros                   SYSTEM TABLE
 D:\tbm\Database\TBM        MSysModules                  SYSTEM TABLE
 D:\tbm\Database\TBM        MSysModules2                 SYSTEM TABLE
 D:\tbm\Database\TBM        MSysObjects                  SYSTEM TABLE
 D:\tbm\Database\TBM        MSysQueries                  SYSTEM TABLE
 D:\tbm\Database\TBM        MSysRelationships            SYSTEM TABLE
 D:\tbm\Database\TBM        MSysToolbars                 SYSTEM TABLE
 D:\tbm\Database\TBM        agence                       TABLE
 D:\tbm\Database\TBM        Chapitre                     TABLE
 D:\tbm\Database\TBM        compteurs                    TABLE
 D:\tbm\Database\TBM        creation                     TABLE
 D:\tbm\Database\TBM        données                      TABLE
 D:\tbm\Database\TBM        ELASTIC                      TABLE
 D:\tbm\Database\TBM        heure contr                  TABLE
 D:\tbm\Database\TBM        QDates                       TABLE
 D:\tbm\Database\TBM        QNum                         TABLE
 D:\tbm\Database\TBM        société                      TABLE
 D:\tbm\Database\TBM        source                       TABLE
 D:\tbm\Database\TBM        Type                         TABLE
 D:\tbm\Database\TBM        ag                           VIEW
 D:\tbm\Database\TBM        compteur                     VIEW
 D:\tbm\Database\TBM        compteurs_Analyse1           VIEW
 D:\tbm\Database\TBM        compteurs_Analyse2           VIEW
 D:\tbm\Database\TBM        compteurs_Analyse4           VIEW
 D:\tbm\Database\TBM        Consultation                 VIEW
 D:\tbm\Database\TBM        donnée                       VIEW
 D:\tbm\Database\TBM        Donnée cumulée agence        VIEW
 D:\tbm\Database\TBM        Donnée cumulée société       VIEW
 D:\tbm\Database\TBM        extrait                      VIEW
 D:\tbm\Database\TBM        Requête1                     VIEW
 D:\tbm\Database\TBM        Requête2                     VIEW
 D:\tbm\Database\TBM        Requête3                     VIEW
 D:\tbm\Database\TBM        Requête4                     VIEW
 D:\tbm\Database\TBM        retours clients              VIEW
 D:\tbm\Database\TBM        SelCAextrait                 VIEW

      'odbc'⎕wi'methods'
 Click Close Create Defer Delete Event Exec Modify New Open
      Ref Send Set SetLinks Close Columns Database Exec Init
       Open Query Tables

      'odbc'⎕wi'Columns' 'Chapitre'
```

```
 Catalog               Scheme Table    Column  ODBC Type
 D:\tbm\Database\TBM          Chapitre Numero  5
 D:\tbm\Database\TBM          Chapitre libelle 12

       Type Name Display Size Buffer Size Digits Radix
       SHORT     5            2           0      10
       TEXT      50           50

       Nullable Remarks
       1        Numéro du chapitre  0
       1        Libellé du chapitre 1

       'odbc'⎕wi'?Exec'
Syntax:  'object'⎕wi'Exec' sql
where  sql  is most any SQL statement (use the Query method for Select s
tatements)
Example:
'odbc'⎕wi'Exec' 'insert into chapitre(numero,libelle) values(199,''BRAVO
'')'
'odbc'⎕wi'Exec' 'delete from chapitre where numero = 199'
'odbc'⎕wi'Exec' 'Update chapitre set libelle=''Provisions'' where numero
 = 199'


       'odbc'⎕wi'Exec' 'select * from chapitre'
  1 Ventes
  2 Activité
  3 Credit clients
  4 Marges
  5 Effectifs
  6 Stocks
  7 Exploitation
  8 Alliance
  9
 10 Quotidien
 99 Chiffres Clés


       'odbc'⎕wi'Exec' 'insert into chapitre(numero,libelle) values(199,'
'BRAVO'')'
0

       'odbc'⎕wi'Exec' 'select * from chapitre'
   1 Ventes
   2 Activité
   3 Credit clients
   4 Marges
   5 Effectifs
   6 Stocks
   7 Exploitation
   8 Alliance
   9
  10 Quotidien
  99 Chiffres Clés
 199 BRAVO

       'odbc'⎕wi'Exec' 'Update chapitre set libelle=''Provisions'' where
numero = 199'
0

       'odbc'⎕wi'Exec' 'select * from chapitre'
   1 Ventes
   2 Activité
   3 Credit clients
   4 Marges
   5 Effectifs
```

```
   6 Stocks
   7 Exploitation
   8 Alliance
   9
  10 Quotidien
  99 Chiffres Clés
 199 Provisions

      'odbc'⎕wi'Exec' 'delete from chapitre where numero = 199'
0

      'odbc'⎕wi'Exec' 'select * from chapitre'
  1 Ventes
  2 Activité
  3 Credit clients
  4 Marges
  5 Effectifs
  6 Stocks
  7 Exploitation
  8 Alliance
  9
 10 Quotidien
 99 Chiffres Clés

      'odbc'⎕wi'Close'
```

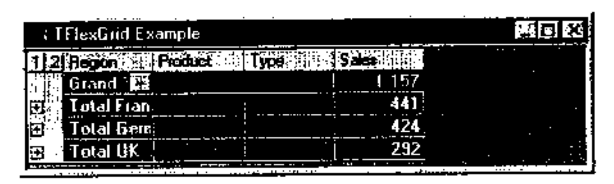
L'objet TflexGrid

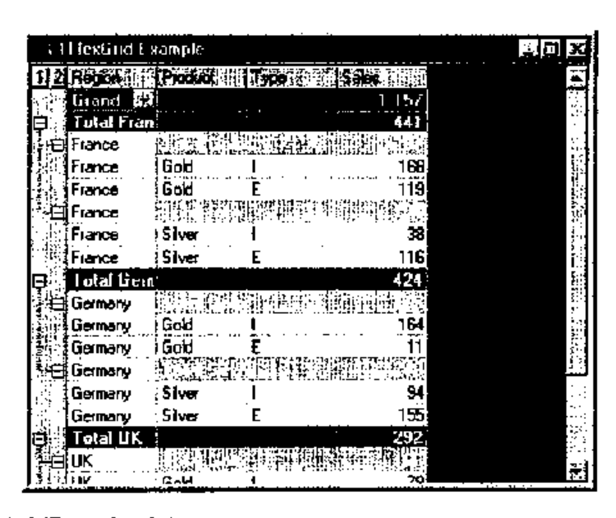
L'objet **TFlexGrid** fait appel à l'objet ActiveX **FlexGrid** de **VideoSoft.**

Il permet d'avoir une grille de saisie hierarchique avec cumuls à plusieurs niveaux et saisie dans les cellules par combo box multi-champs.

Exemple:

```
      FlexGridExemple
```

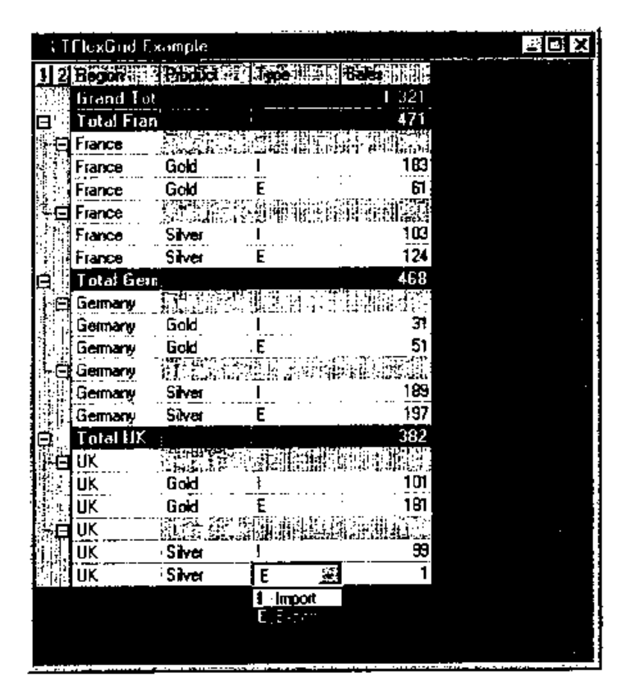





```
    ∇ FlexGridExemple;data
      ⍝∇ FlexGridExemple -- Exemple montrant l'objet TFlexGrid
      ⍝∇ Nécessite l'objet ActiveX VSFLEX6.OCX de Videosoft installé

      data←(4/'France' 'Germany' 'UK'),(12⍴2/'Gold' 'Silver')
      data←data,(12⍴'I' 'E'),[1.5]⍕¨?12⍴200
      data←'Region' 'Product' 'Type' 'Sales'⍪data
      ⎕wself←'fmFG'⎕wi'*Create' 'TForm'('caption' 'TFlexGrid Example')
      ⎕wself←'fmFG.ss'⎕wi'*Create' 'TFlexGrid'
          ⎕wi'wherelc'0 0 ¯300 ¯400
          ⎕wi'attach'1 2 3 4
          ⎕wi'allowediting'1
          ⎕wi'allowsortcols'1
          ⎕wi'allowmovecols'1
          ⎕wi'borderstyle'1
          ⎕wi'Rows'1
          ⎕wi'Cols'1
          ⎕wi'FixedRows'1
          ⎕wi'FixedCols'1
          ⎕wi'ColWidth'0 300
          ⎕wi'Add'(0 1)data
          ⎕wi'colcombo'1'France|Germany|UK'
          ⎕wi'colcombo'2'Gold|Silver'
          ⎕wi'colcombo'3'I;Import|E;Export'
      ⍝ Argument de OutlineBar:
      ⍝ 0=sans boutons 1=complète 2=sans boutons du haut
      ⍝ 3=sasn boutons du haut et sans lignes de connexion
          ⎕wi'OutlineBar'1
      ⍝ SubTotalPosition: 0=totaux en bas  1=totaux en haut
          ⎕wi'SubTotalPosition'1
          ⎕wi'SubTotal' 'Sum'4(¯1 1 2)
      'fmFG'⎕wi'*size'300 400
      'fmFG'⎕wi'DemoShow'
      ⎕wself←'fmFG.ss'
    ∇
```

La fonction **FlexGridExemple** montre l'utilisation de l'objet **TFlexGrid** ainsi que de certaines de ses propriétés et méthodes.



L'affichage précédent montre la saisie en cours dans le champ **Type** par utilisation d'une Combo multi-champs.

Some Other Triangles and APL

Joseph De Kerf

Abstract: In this provisionally last paper of the series (1)(2)(3)(4), we treat "some other" triangles. Properties and interconnections are discussed. APL defined functions for generating those triangles are given. Results of benchmarks on cpu time are reported. It may be worthwhile to note that in general, algorithms based on recursion, are more efficient in cpu time than those based on the closed forms.

∴

0. Introduction

In a series of previous papers we treated the Pascal triangle (1), the De Moivre triangles (2), the Hoggatt triangles(3), and the Horadam triangles (4). In this provisionally last paper of the series, we treat "some other" triangles. Some details about triangles may be found in B.A. Bondarenko (5) and in the more recent book published by J.A. Conway and R.K. Guy (6).

All defined functions shown generate the triangles as a nested vector of vectors containing the N first rows of the triangles, such that the form

```
⊃⍕¨F N
```

transforms the nested vector of vectors generated by F to a simple character matrix containing the N rows of the triangle, adjusted to the left and padded with blanks up to the character length of the last row. The defined function CENTER rotates the rows of this character matrix such that those rows are centered (4).

The function CENTER has been taken over from those published in the book "APL2 in Depth", by N.D. Thomson and R.P. Polivka (7). Defined functions shown conform to the proposed new standard ISO APL-Extend-

```
      ∇CENTER[⎕]∇
     ∇Z←CENTER M;R
[1]   R←+/∧\' '=⌽M
[2]   Z←(-⌊0.5×R)⌽M
     ∇
```

ed, prepared by the ISO APL Working Group ISO-IEC/JTC1/SC22/WG3 (21) Benchmarks have been done on a MicroLine Pentium S-100, with mathematical co-processor, using APL/W Version 7.1.2, under Windows 3.11 (22). Migration Level ⎕ML was set to 2, the default value value being 0. CPU times have been measured using the system function ⎕MONITOR.

<u>Note</u>: In some illustrations, a Generalized Fibonacci sequence, what we call an Horadam sequence, is generated. The defined function HORS is used. The definition of HORS may be found in (4).

1. <u>The Bell Triangle</u>

In 1980, J. Shallit (8) defined what he called the Bell triangle, a triangle B(n,k) with k = 1(1)n, as generated by the recursion algorithm:

$$B(1,1) = 1$$
$$B(n,1) = B(n-1,n-1) \qquad (2 \leq n)$$
$$B(n,k) = B(n,k-1) + B(n-1,k-1) \qquad (2 < k \leq n)$$

The inconveniess of this algorithm is that, for calculating B(n,k), the predecessor B(n,k-1) in it's row must be known. This inconvenience may be removed by transforming the algorith to the form (9):

$$B(1,1) = 1$$
$$B(n,1) = B(n-1,n-1) \qquad (2 \leq n)$$
$$B(n,k) = B(n-1,n-1) + \sum_{r=1}^{k-1} B(n-1,r) \qquad (2 \leq k < n)$$

such that the elements of the nth row B(n,k) may be calculated from the elements of the previous row only.

A defined function BELLTR1 generating the Bell triangle, based on the algorithm in it's transformed form, is for instance:

```
      ∇BELLTR1[⎕]∇
     ∇Z←BELLTR1 N
[1]   →(N=1)/0,Z←1⍴⊂,1
[2]   Z←BELLTR1 N-1
[3]   Z←Z,⊂(↑⌽↑⌽Z)+0,+\⊃↑⌽Z
     ∇
      CENTER ⊃⍕¨BELLTR1 8
                 1
                1 2
               2 3 5
             5 7 10 15
           15 20 27 37 52
        52 67 87 114 151 203
    203 255 322 409 523 674 877
877 1080 1335 1657 2066 2589 3263 4140
```

The nth row contains n elements. The left side of the triangle, from top to down, is the Bell sequence $B_0, B_1, B_2, \ldots$ and the right side of the triangle, from top to down, is the Bell sequence B_1, B_2, B_3, ... Rows are not symmetric, the nth row being a strict ascending sequence with as first element B_{n-1} and as last element B_n. This means that the maximum of the domain of the right argument of BELLTR1 is N = 218, as B_{218} = 6.101309834E306 and B_{219} exceeds the largest number representable 1.797693135E308.

```
      ↑¨BELLTR1 10+1
1 1 2 5 15 52 203 877 4140 21147 115975
      1,↑¨⌽¨BELLTR1 10
1 1 2 5 15 52 203 877 4140 21147 115975
```

Shallit reported several properties of the elements of the triangle, for instance that the sum of the elements of the nth row is the difference $B_{n+1} - B_n$:

```
      +/¨BELLTR1 10
1 3 10 37 151 674 3263 17007 94828 562595
      (2↓↑¨BELLTR1 12)-1↓↑¨BELLTR1 11
1 3 10 37 151 674 3263 17007 94828 562595
      (1↓↑¨⌽¨BELLTR1 11)-↑¨⌽¨BELLTR1 10
1 3 10 37 151 674 3263 17007 94828 562595
```

BELLTR1 uses implicit recursion. A defined function BELLTR2, based

on the same algorithm, but using a programmed recursion loop, may be

```
      ∇BELLTR2[⎕]∇
     ∇Z←BELLTR2 N
[1]   →(N=1)/0,Z←1⍴⊂,1
[2]   END:Z←Z,⊂(↑⌽↑⌽Z)+0,+\⊃↑⌽Z
[3]   →(N>⍴Z)/END
     ∇
```

Benchmarks on cpu time have been done for N = 40(40)200. Results are shown below (in ms):

N	40	80	120	160	200
BELLTR1 N	26	64	119	195	287
BELLTR2 N	17	46	91	157	240

As N increases, BELLTR2 is from 53% to 20% more efficient in cpu time than BELLTR1, rather considerable, even for higher values of N.

The question raises, if generating the Bell sequence $B_0,B_1,B_2,...$ from BELLTR1 or BELLTR2 as shown above, may compete with explicit algorithms. The process may be considerably accelerated by incorporating the deduction of the Bell numbers from the rows of the Bell triangle in the defined functions themselves, such as is done in the defined functions BELLA and BELLB, respectively using implicit recursion and a programmed recursion loop (cfr next page).

Benchmarks on cpu time have been done for N = 40(40)200. Results are shown below (in ms):

N	40	80	120	160	200
BELLA N	18	39	62	89	118
BELLB N	13	28	46	67	91

As N increases, BELLB is from 38% to 30% more effincient in cpu time than BELLA, still rather considerable, even for higher values of N.

```
      ∇BELLA[⎕]∇
     ∇Z←BELLA N
[1]   →(N≤1)/0,Z←(1+N⌊1)⍴R←1
[2]   Z←BELLA N-1
[3]   Z←Z,¯1↑R←(¯1↑R)+0,+\R
     ∇
      BELLA 10
1 1 2 5 15 52 203 877 4140 21147 115975

      ∇BELLB[⎕]∇
     ∇Z←BELLB N;R
[1]   →(N≤1)/0,Z←(1+N⌊1)⍴R←1
[2]   END:Z←Z,¯1↑R←(¯1↑R)+0,+\R
[3]   →(N≥⍴Z)/END
     ∇
      BELLB 10
1 1 2 5 15 52 203 877 4140 21147 115975
```

Most important however, as shown in (9), the algorithm is much more efficient in cpu time than all other algorithms investigated in that paper. Apparently this is mainly due to the fact the algorithm only uses addition. It is doubtful if a considerable more efficient algorithm may be found.

Note: In 1934, E.T. Bell (10) introduced what he called the exponential numbers $B_0, B_1, B_2, \ldots$ as those defined by the Maclaurin expansion of the generating function (e*e*x)÷e:

$$e^{e^x} = e \sum_{n=o}^{\infty} B_n \frac{x^n}{!n}$$

$$= e \left(B_0 + B_1 \frac{x}{!1} + B_2 \frac{x^2}{!2} + B_3 \frac{x^3}{!3} + \ldots\right)$$

There is a strong relation between those Bell numbers and the Stirling set numbers or Stirling numbers of the second kind $S_2(n,k)$. He showed that:

$$B_n = \sum_{k=1}^{n} S_2(n,k)$$

$$\text{with } S_2(n,k) = \frac{1}{!(k-1)} \left[\sum_{r=0}^{k-1} (-1)^r \binom{k-1}{r} (k-r)^{n-1} \right]$$

A defined function BELLS, based on this algorithm, returning as explicit result the Bell number B_n, is for instance:

```
      ∇BELLS[⎕]∇
     ∇Z←BELLS N;K;R
[1]   →(+/N=0 1)/0,Z←K←1
[2]   END:R←¯1+⍳K←K+1
[3]   Z←Z+(-/(R!K-1)×(K-R)*N-1)÷!K-1
[4]   →(K<N)/END
     ∇
      BELLS 0
1
      BELLS 10
115975
      BELLS¨0,⍳10
1 1 2 5 15 52 203 877 4140 21147 115975
```

The maximum of the domain of the right argument N is N = 137, with B_{137} = 1.365939847E172, as for N = 138 the alternating sum, before being divided by !K-1, exceeds the largest number representable 1.797693135E308. Benchmarks on cpu time have been done for N = 40 (40)120. Results are shown below (in ms):

N	40	80	120
BELLS	34	127	311
BELLS¨0,⍳N	551	3536	11967

As N increases, the form BELLS¨0,⍳N is from 42 to 260 times slower than the form BELLS N. The algorithm is only from theoretical importance, but is unsuited for generating the Bell numbers.

2. The Stirling Triangle of the First Kind

The Stirling cycle numbers or Stirling numbers of the first kind $S_1(n,k)$ are the number of permutations of n objects that have just k cycles (k = 1...n). The Stirling set numbers or Stirling numbers of the second kind $S_2(n,k)$ are the number of groupings of n distinct objects into exactly k groups (k = 1...n). A lot of details

about Stirling numbers may be found in (11). Tables of Stirling numbers have been published in (12). Stirling triangles of the first and second kind are defined as those triangles whose nth row is the Stirling sequence $S_1(n,k)$ or $S_2(n,k)$ respectively.

The Stirling numbers of the first kind $S_1(n,k)$ may be calculated using the recursion algorithm:

$S_1(1,1) \;=\; 1$

$S_1(n,k) \;=\; S_1(n-1),k-1) + (n-1)S_1(n-1,k)$

A defined function STFKTR1 generating the Stirling triangle of the first kind, based on this algorithm, is for instance:

```
      ∇STFKTR1[⎕]∇
     ∇Z←STFKTR1 N
[1]   →(N≤1)/0,Z←(N⌊1)⍴⊂⍳1
[2]   Z←STFKTR1 N-1
[3]   Z←Z,⊂(0,↑⌽Z)+((⍴↑⌽Z)×↑⌽Z),0
     ∇
      CENTER ⊃⍕¨STFKTR1 8
                 1
                1 1
               2 3 1
             6 11 6 1
           24 50 35 10 1
        120 274 225 85 15 1
     720 1764 1624 735 175 21 1
5040 13068 13132 6769 1960 322 28 1
```

The nth row contains n elements. Rows are not symmetric. The first element of the nth row is !(n-1). The sum of the elements of the nth row is !n. For $n > 1$, the alterneting sum of the elements of the rows is 0. The maximum of the domain of the right argument of STFKTR1 is N = 170, as $S_1(170,5)$ = 1.421086266E306 and $S_1(171,5)$ exceeds the largest number representable 1.797693135E308.

```
      ↑¨STFKTR1 10
1 1 2 6 24 120 720 5040 40320 362880
      !0,⍳9
1 1 2 6 24 120 720 5040 40320 362880
```

```
      +/¨STFKTR1 10
1 2 6 24 120 720 5040 40320 362880 3628800
      !⍳10
1 2 6 24 120 720 5040 40320 362880 3628800
      -/¨STFKTR1 10
1 0 0 0 0 0 0 0 0 0
```

STFKTR1 uses implicit recursion. A defined function STFKTR2, based on the same algorithm, but using a programmed recursion loop, may be

```
       ∇STFKTR2[⎕]∇
      ∇Z←STFKTR2 N
[1]    →(N≤1)/0,Z←(N⌊1)⍴⊂⍳1
[2]    END:Z←Z,⊂(0,↑⌽Z)+((⍴↑⌽Z)×↑⌽Z),0
[3]    →(N>⍴Z)/END
      ∇
```

Benchmarks on cpu time have been done for N = 40(40)160. Results are shown below (in ms):

N	40	80	120	160
STFKTR1 N	27	66	126	211
STFKTR2 N	17	48	100	175

As N increases, STFKTR2 is from 59% to 21% more efficient in cpu time than STFKTR1, rather considerable, even for higher values of N.

<u>Note</u>: In some publications, the Stirling numbers of the first kind are defined by the recursion algorithm:

$S_1(1,1) = 1$

$S_1(n,k) = S_1(n-1,k-1) - (n-1)S_1(n-1,k)$

The effect of this is, that the numbers as defined in the text, are multiplied by a factor $(-1)^{n-k}$. In other words, if n is odd and k is even or if n is even and k is odd, the numbers are given a negative sign.

The corresmonding triangle becomes:

```
                 1
               ¯1 1
              2 ¯3 1
            ¯6 11 ¯6 1
          24 ¯50 35 ¯10 1
      ¯120 274 ¯225 85 ¯15 1
    720 ¯1764 1624 ¯735 175 ¯21 1
¯5040 13068 ¯13132 6769 ¯1960 322 ¯28 1
```

such that for $n > 1$, the sum of the elements of the rows, instead of their alternating sum, is 0. On the other hand, the algorithm is in contradiction with the physical definition of the Stirling numbers of the first kind.

3. The Stirling Triangle of the Second Kind

The Stirling numbers of the second kind $S_2(n,k)$ may be calculated from the recursion algorithm:

$$S_2(1,1) = 1$$
$$S_2(n,k) = S_2(n-1,k-1) + kS_2(n-1,k)$$

A defined function STSKTR1 generating the Stirling triangle of the second kind, based on this algorithm, is for instance:

```
      ∇STSKTR1[⎕]∇
     ∇Z←STSKTR1 N
[1]   →(N≤1)/0,Z←(N⌊1)⍴⊂⍳1
[2]   Z←STSKTR1 N-1
[3]   Z←Z,⊂(0,↑⌽Z)+((⍳⍴↑⌽Z)×↑⌽Z),0
     ∇
      CENTER ⊃⍕¨STSKTR1 8
              1
             1 1
            1 3 1
           1 7 6 1
         1 15 25 10 1
       1 31 90 65 15 1
    1 63 301 350 140 21 1
1 127 966 1701 1050 266 28 1
```

The nth row contains n elements. Rows are not symmetric. As already

mentioned in the first section of this paper, the sum of the elements of the nth row is the Bell number B_n. The maximum of the domain of the right argument of STSKTR1 is N = 219, as $S_2(219,54)$ = 4.190119214E307 and $S_2(220,54)$ exceeds the largest number representable 1.797693135E308.

```
      +/¨STSKTR1 10
1 2 5 15 52 203 877 4140 21147 115975
      1↓BELLB 10
1 2 5 15 52 203 877 4140 21147 115975
```

STSKTR1 uses implicit recursion. A defined function STSKTR2, based on the same algorithm, but using a programmed recursion loop, may be

```
      ∇STSKTR2[⎕]∇
     ∇Z←STSKTR2 N
[1]   →(N≤1)/0,Z←(N⌊1)⍴⊂⍳1
[2]   END:Z←Z,⊂(0,↑⌽Z)+((⍳⍴↑⌽Z)×↑⌽Z),0
[3]   →(N>⍴Z)/END
     ∇
```

Benchmarks on cpu time have been done for N = 40(40)200. Resulats are shown below (in ms):

N	40	80	120	160	200
STSKTR1 N	27	70	136	227	381
STSKTR2 N	18	50	105	186	320

As N increases, STSKTR2 is from 50% to 19% more efficient in cpu time than STSKTR1, rather considerable, even for higher values of N.

<u>Note</u>: The Stirling numbers of the first kind $S_1(n,k)$ and the Stirling numbers of the second kind $S_2(n,k)$ may be calculated from the closed forms:

$$S_1(n,k) = \sum_{r=1}^{n-k} (-1)^{n-k+r} \binom{n-1+r}{n-k+r} \binom{2n-k}{n-k-r} S_2(n-k+r,r)$$

$$S_2(n,k) = \frac{1}{!(k-1)} \left[\sum_{r=0}^{k-1} (-1)^r \binom{k-1}{r} (k-r)^{n-1} \right]$$

As n increases, accuracy is degenerating. In section one of this paper, it was shown that a defined function based on the second form is very slow. A fortiori for the first form. The algorithms are only from theoretical importance, but are unsuited for the generation of the Stirling sequences and the Stirling triangles.

4. The Eulerian Triangle

The Eulerian numbers E(n,k) are the number of permutations of ⍳n which have exactly k ascending runs (k = 1...n). They should be distinguished from the so-called Euler numbers E_n, which occur as coefficients in the power series expansions of the secant and hyperbolic secant functions. A lot of details about Eulerian numbers may be found in (11). We define in this paper the Eulerian triangle as the triangle whose nth row is the Eulerian sequence E(n,k).

The Eulerian sequence E(n,k) may be calculated from the recursion algorithm:

$E(1,1) = 1$

$E(n,k) = (n-k+1)E(n-1,k-1) + kE(n-1,k)$

A defined function EULRTR1 generating the Eulerian triangle, based on this algorithm, is for instance:

```
      ∇EULRTR1[⎕]∇
     ∇Z←EULRTR1 N
[1]   →(N≤1)/0,Z←(N⌊1)⍴⊂⍳1
[2]   Z←EULRTR1 N-1
[3]   Z←Z,⊂((⌽⍳1+⍴↑⌽Z)×0,↑⌽Z)+((⍳⍴↑⌽Z)×↑⌽Z),0
     ∇
```

The nth row contains n elements. Rows are symmetric. The sum of the elements of the nth row is !n. The maximum of the domain of the right argument of EULRTR1 is N = 171, as E(171,86) = 1.306578448E308 and E(172,86) = E(172,87) exceeds the largest number representable 1.797693135E308.

```
      CENTER ⊃⍕¨EULRTR1 8
               1
              1 1
             1 4 1
           1 11 11 1
         1 26 66 26 1
       1 57 302 302 57 1
  1 120 1191 2416 1191 120 1
1 247 4293 15619 15619 4293 247 1

      +/¨EULRTR1 10
1 2 6 24 120 720 5040 40320 362880 3628800
      !⍳10
1 2 6 24 120 720 5040 40320 362880 3628800
```

It may be shown that (11): $\sum_{k=1}^{n} \binom{m+k-1}{n} E(n,k) = m^n$

such that for instance, if m = 5 and n = 10, the sum is 5^{10}:

```
      +/(10!¯1+5+⍳10)×↑⌽EULRTR1 10
9765625
      5*10
9765625
```

EULRTR1 uses implicit recursion. A defined function EULRTR2, based on the same algorithm, but using a programmed recursion loop, may be

```
      ∇EULRTR2[⎕]∇
     ∇Z←EULRTR2 N
[1]   →(N≤1)/0,Z←(N⌊1)⍴⊂⍳1
[2]   END:Z←Z,⊂((⌽⍳1+⍴↑⌽Z)×0,↑⌽Z)+((⍳⍴↑⌽Z)×↑⌽Z),0
[3]   →(N>⍴Z)/END
     ∇
```

Benchmarks on cpu time have been done for N = 40(40)160. Results are shown on next page (in ms).

As N increases, EULRTR1 is from 43% to 17% more efficient in cpu time than EULRTR1, rather considerable, even for higher values of N.

N	40	80	120	160
EULRTR1 N	33	83	161	260
EULRTR2 N	23	64	132	223

Note: The Eulerian numbers E(n,k) may be calculated from the closed form (k = 1...n):

$$E(n,k) = \sum_{r=0}^{k} (-1)^r \binom{n+1}{r} (k-r)^n$$

As n increases, accuracy is degenerating. This degeneration may be reduced to some extent, by only calculating E(n,k) for k ≤ ⌈n+2, and catenating the result with the reverse of this result for k ≤ ⌊n+2. In addition, apart of some overhead, the efficiency in cpu time is improved with a factor of about 2.

A defined function EULRS, based on this algorithm, returning as explicit result the sequence E(n,k) for k = 1(1)n, may be:

```
      ∇EULRS[⎕]∇
     ∇Z←EULRS N;K;R
[1]   →(N≤1)/0,Z←K←1
[2]   END:R←0,⍳K←K+1
[3]   Z←Z,-/⌊0.5+(R!N+1)×(K-R)*N
[4]   →(K<⌈N÷2)/END
[5]   Z←Z,⌽(⌊N÷2)↑Z
     ∇
      EULRS 10
1 1013 47840 455192 1310354 1310354 455192 47840 1013 1
      ↑⍎EULRTR2 10
1 1013 47840 455192 1310354 1310354 455192 47840 1013 1
```

The maximum of the domain of the right argument is N = 158, giving E(158,79) = E(158,80) = 3.98881832E291, as for N = 159 some of the terms of the alternating sum exceed in absolute value the largest number representable 1.797693135E308.

Benchmarks on cpu time have been done for N = 40(40)120. Results are shown on next page (in ms).

N	40	80	120
EULRS N	17	62	159
EULRS¨⍳N	282	1786	6095

As N increases, the form EULRS¨⍳N is from 12 to 46 times slower than the form EULRTR2 N. The algorithm is only from theoretical importance,. but unsuited for the generation of the Eulerian sequences and the Eulerian triangle.

5. The Wong and Maddockx Triangle

In 1975, C.K. Wong and T.W. Maddockx studied the numbers M(n,k) satisfying the recursion relation (13):

M(1,1) = M(2,1) = M(2,2) = 1

M(n,k) = M(n-2,k-1) + M(n-1,k-1) + M(n-1,k)

We call in this paper the Wong and Maddockx triangle, the triangle whose nth row is the Wong and Maddockx sequence M(n,k) with k = 1...n. A defined function WOMATR1, generating the Wong and Maddockx triangle, based on the recursion algorithm shown, is for instance:

```
      ∇WOMATR1[⎕]∇
     ∇Z←WOMATR1 N
[1]   →(N≤1)/0,Z←(N⌊1)⍴⊂⍳1
[2]   →(N=2)/0,Z←Z,⊂2⍴1
[3]   Z←WOMATR1 N-1
[4]   Z←Z,⊂(1,(↑¯2↑Z)+(¯1↓↑⌽Z)+1↓↑⌽Z),1
     ∇
      CENTER ⊃⍕¨WOMATR1 8
          1
         1 1
        1 3 1
       1 5 5 1
     1 7 13 7 1
    1 9 25 25 9 1
  1 11 41 63 41 11 1
1 13 61 129 129 61 13 1
```

The nth row contains n elements. Rows are symmetric. The sum of the

elements of the nth row is the Pell number P_n (4). The maximum of the domain of the right argument is N = 810, as M(810,405) = M(810,406)= 1.316673461E308, and M(811,406) exceeds the largest number representable 1.797693135E308.

```
      +/¨WOMATR1 10
1 2 5 12 29 70 169 408 985 2378
      P←1 2 1 2
      P HORS 10
1 2 5 12 29 70 169 408 985 2378
```

Wong and Maddockx also showed that the sums of the elements on the ascending diagonals of the triangle form the Tribonacci sequence: 1,1,2,4,... .

```
      10↑+/(1-⍳10)⌽(⊃WOMATR1 10),[1]9 10⍴0
1 1 2 4 7 13 24 44 81 149
```

WOMATR1 uses implicit recursion. A defined function WOMATR2, based on the same algorithm, but using a programmed recursion loop, may be

```
      ∇WOMATR2[⎕]∇
     ∇Z←WOMATR2 N
[1]   →(N≤1)/0,Z←(N⌊1)⍴⊂⍳1
[2]   →(N=2)/0,Z←Z,⊂2⍴1
[3]   END:Z←Z,⊂(1,(↑¯2↑Z)+(¯1↓↑⌽Z)+1↓↑⌽Z),1
[4]   →(N>⍴Z)/END
     ∇
```

Benchmarks on cpu time have been done for N = 100(100)500. Results are shown below (in ms):

N	100	200	300	400	500
WOMATR1 N	113	320	632	1060	1675
WOMATR2 N	70	233	503	879	1447

The algorithm looks relatively slow, but for N = 200, the cpu times are of the same order as those for the other triangles treated in

this paper. As N increases, WOMATR2 is from 61% to 16% more efficient in cpu time than WOMATR1, rather considerable, even for higher values of N.

Note: Tribonacci sequences $T_1, T_2, T_3, \ldots$ may be defined by the recursion algorithm:

$$T_1 = p$$
$$T_2 = q$$
$$T_3 = r$$
$$T_n = aT_{n-3} + bT_{n-2} + cT_{n-1} \qquad (n > 3)$$

the sequence generated by the sums of the elements on the ascending diagonals of the Wong and Maddockx triangle being the Tribonacci sequence $p = 1 \quad q = 1 \quad r = 2 \quad a = 1 \quad b = 1 \quad c = 1$.

Defined functions TRIB1 and TRIB2 generating the Tribonacci sequence, based on this algorithm, and using implicit recursion or a programmed recursion loop respectively, are for instance:

```
      ∇TRIB1[⎕]∇
     ∇Z←P TRIB1 N
[1]   →(N≤3)/0,Z←(N⌊3)↑P
[2]   Z←P TRIB1 N-1
[3]   Z←Z,+/(¯3↑P)×¯3↑Z
     ∇
      ∇TRIB2[⎕]∇
     ∇Z←P TRIB2 N
[1]   →(N≤3)/0,Z←(N⌊3)↑P
[2]   END:→(N>⍴Z←Z,+/(¯3↑P)×¯3↑Z)/END
     ∇
```

the left argument P being the vector p,q,r,a,b,c and the right argument N specifying the number of elements of the sequence requested.

```
      P←1 1 2 1 1 1
      P TRIB1 10
1 1 2 4 7 13 24 44 81 149
      P TRIB2 10
1 1 2 4 7 13 24 44 81 149
```

The Tribonacci numbers T_n may also be calculated explicitly from the

auxiliary equation $t^3 - ct^2 - bt - a = 0$. The question raises whether, just like for the Horadam sequences, calculating the Tribonacci sequence directly from the auxiliary equation is much more efficient in cpu time than generating the sequence from the recursion algorithm (14). A paper on Tribonacci sequences and a paper on Tribonacci triangles will be submitted for publication in the near future.

6. The Shapiro Triangle

In 1976, L.W. Shapiro constructed the triangle whose elements of the nth row are the sequence $S(n,k)$ with $k = 1...n$, sequence defined by the recursion algorithm (15):

$$S(1,1) = 1$$

$$S(n,k) = S(n-1,k-1) + 2S(n-1,k) + S(n-1,k+1)$$

A defined function SHAPTR1, generating the Shapiro triangle, based on this recursion algorithm, is for instance:

```
      ∇SHAPTR1[⎕]∇
     ∇Z←SHAPTR1 N
[1]   →(N≤1)/0,Z←(N⌊1)⍴⊂⍳1
[2]   Z←SHAPTR1 N-1
[3]   Z←Z,⊂((0,¯1↓↑⌽Z)+(2×↑⌽Z)+(1↓↑⌽Z),0),1
     ∇

      CENTER ⊃⍕¨SHAPTR1 8
              1
             2 1
            5 4 1
          14 14 6 1
         42 48 27 8 1
     132 165 110 44 10 1
   429 572 429 208 65 12 1
1430 2002 1638 910 350 90 14 1
```

The nth row contains n elements. Rows are not symmetric. The first element of the nth row is the Catalan number C_n, such that the left side of the triangle, from top to down, is the Catalan sequence $C_1,C_2,C_3,...$. The sum of the elements of the nth row is

$\frac{1}{2}$ (n+1)C_n. The maximum of the domain of the right argument N is 517, as S(517,16) = 8.61680321E307, and S(218,16) exceeds the largest number representable 1.797693135E308.

```
      ↑¨SHAPTR1 10
1 2 5 14 42 132 429 1430 4862 16796
      +/¨SHAPTR1 10
1 3 10 35 126 462 1716 6435 24310 92378
      0.5×(1+⍳10)×↑¨SHAPTR1 10
1 3 10 35 126 462 1716 6435 24310 92378
```

Shapiro showed that the numbers S(n,k) may be expressed as a sum of products of Catalan numbers:

$S(n,k) = \sum C_{i_1} C_{i_2} C_{i_3} \ldots C_{i_k}$ where the summation is over all values for which $i_1 + i_2 + i_3 + \ldots + i_k = n$. This is why Shapiro called his construction a Catalan triangle.

SHAPTR1 uses implicit recursion. A defined function SHAPTR2, based on the same algorithm, but using a programmed recursion loop, may be

```
      ∇SHAPTR2[⎕]∇
     ∇Z←SHAPTR2 N
[1]   →(N≤1)/0,Z←(N⌊1)⍴⊂⍳1
[2]   END:Z←Z,⊂((0,¯1↓↑⌽Z)+(Z×↑⌽Z)+(1↓↑⌽Z),0),1
[3]   →(N>⍴Z)/END
     ∇
```

Benchmarks on cpu time have been done for N = 100(100)500. Results are shown below (in ms):

N	100	200	300	400	500
SHAPTR1 N	111	352	731	1253	2003
SHAPTR2 N	87	303	658	1158	1876

The algorithm looks relatively slow, but for N = 200, the cpu times are of the same order as those for the other triangles treated in this paper. SHAPTR2 is 28% and 16% more efficient in cpu time than

SHAPTR1 for N = 100 and N = 200 respectively, rather considerable for that domain of N.

Note: The Catalan number C_n is the number of trees with n nodes. It may be shown that (11):

$$C_n = \frac{1}{n+1} \binom{2n}{n}$$

A defined function CATAS, based on this form, generating a vector containing the Catalan sequence $C_1, C_2, C_3, \ldots$, the right argument N specifying the number of elements of the sequence requested, is for instance:

```
      ∇CATAS[⎕]∇
     ∇Z←CATAS N
[1]   Z←(N!2×N)÷1+N←⍳N
     ∇
      CATAS 10
1 2 5 14 42 132 429 1430 4862 16796
      0.5×(1+⍳10)×CATAS 10
1 3 10 35 126 462 1716 6435 24310 92378
```

Benchmarks on cpu time have been done for N = 100(100)500. Results are shown below (in ms):

N	100	200	300	400	500
CATAS N	4.5	13.1	27.9	46.6	71.4

CPU times are very low. It is doubtful if a considerable more efficient algorithm may be found.

7. The Agarwal Triangle

In 1990, A.K. Agarwal introduced a new class of numbers A(n,k), defined by (16):

$$A(n,k) = (-1)^{n-k} \binom{2n-1}{n-k} L_{2k-1} \qquad (1 \leq k \leq n)$$

L_{2k-1} being the Lucas number of order 2k-1 (4). We call in this pa-

per the Agarwal triangle, the triangle whose nth row is the Agarwal sequence A(n,k) with k = 1...n .

Lucas numbers L_n are the Horadam numbers H_n, with parameters p = 1 q = 3 r = 1 s = 1 . In (14), it is shown that, calculating the Horadam numbers explicitly from the auxiliary equation, this means by the Binet form, is much more efficient in cpu time than calculating those numbers from the recursion algorithm. For the Fibonacci numbers F_n and the Lucas numbers L_n, this Binet form is respectively (17):

$$F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

= 1 1 2 3 5 8 13 21 34 55 ...

$$L_n = \left(\frac{1+\sqrt{5}}{2}\right)^n + \left(\frac{1-\sqrt{5}}{2}\right)^n$$

= 1 3 4 7 11 18 29 47 76 123 ...

It has been shown that (18), for the calculation of the Fibonacci numbers F_n, as the second term converges to 0, it suffices to calculate the first term and rounding off the result to an integer. Apart of some overhead, the calculation is speeded up by a factor of about 2. For n > 1, the same process may be applied when calculating the Lucas numbers L_n:

n:	$\left(\frac{1+\sqrt{5}}{2}\right)^n$	rounded
1	1.61803	-
2	2.61803	3
3	4.23607	4
4	6.85410	7
5	11.09017	11
6	17.94427	18
7	29.03444	29
8	46.97871	47
9	76.01316	76
10	122.99187	123
..	....	...

A defined function AGARTR1, generating the Agarwal triangle, based

on the definition given above, and incorporating the shortcut specified, is for example:

```
      ∇AGARTR1[⎕]∇
     ∇Z←AGARTR1 N;K;L;M
[1]   →(N≤1)/0,Z←(N⌊1)⍴⊂⍳1
[2]   L←1,⌊0.5+(0.5×1+5*0.5)*1+2×⍳N-1
[3]   Z←AGARTR1 N-1
[4]   K←⍳M←1+⍴↑⌽Z
[5]   Z←Z,⊂(¯1*M-K)×(M↑L)×(M-K)!¯1+2×M
     ∇

      CENTER ⊃⍕¨AGARTR1 8
                        1
                       ¯3 4
                    10 ¯20 11
                  ¯35 84 ¯77 29
               126 ¯336 396 ¯261 76
          ¯462 1320 ¯1815 1595 ¯836 199
      1716 ¯5148 7865 ¯8294 5928 ¯2587 521
¯6435 20020 ¯33033 39585 ¯34580 20895 ¯7815 1364
```

The nth row contains n elements. Rows are not symmetric. If n-k is even, A(n,k) is positive. If n-k is odd, A(n,k) is negative. The sum of the elements of the rows is 1. The maximum of the domain of the right argument is N = 443, as A(443,199) = 5.900204722E307, and the absolute value of A(444,199) exceeds the largest number representable 1.797693135E308.

```
      +/¨AGARTR1 10
1 1 1 1 1 1 1 1 1 1
```

The absolute value of the first element of the nth row of the triangle is $\frac{1}{2}$ (n+1) C_n, and as such is the sum of the elements of the nth row of the Shapiro triangle.

```
      |↑¨AGARTR1 10
1 3 10 35 126 462 1716 6435 24310 92378
      0.5×(1+⍳10)×CATAS 10
1 3 10 35 126 462 1716 6435 24310 92378
      +/¨SHAPTR2 10
1 3 10 35 126 462 1716 6435 24310 92378
```

The last or nth element of the nth row of the triangle A(n,n) is the Lucas number L_{2n-1}.

```
      ↑"⍕"AGARTR1 10
1 4 11 29 76 199 521 1364 3571 9349
      P←1 3 1 1
      (19⍴1 0)/P HORS 19
1 4 11 29 76 199 521 1364 3571 9349
```

AGARTR1 uses implicit recursion. A defined function AGARTR2, based on the same algorithm, but using a programmed recursion loop, may be

```
      ∇AGARTR2[⎕]∇
     ∇Z←AGARTR2 N;K;L;M
[1]   →(N≤1)/0,Z←(N⌊1)⍴⊂⍳1
[2]   L←1,⌊0.5+(0.5×1+5*0.5)*1+2×⍳N-1
[3]   END:K←⍳M←1+⍴↑⌽Z
[4]   Z←Z,⊂(¯1*M-K)×(M↑L)×(M-K)!¯1+2×M
[5]   →(N>⍴Z)/END
     ∇
```

Benchmarks on cpu time have been done for N = 100(100)400. Results are shown below (in ms):

N	100	200	300	400
AGARTR1 N	509	2967	9056	20505
AGARTR2 N	417	2713	8560	19686

The algorithm is very slow: for N = 400 the cpu time is about 20 seconds. Agarwal however showed that for $k<n$, the element A(n,k) may be calculated from the element A(n-1,k) through the recursion algorithm:

$$A(1,1) = 1$$

$$A(n,k) = -\frac{2(n-1)(2n-1)}{(n-k)(n+k-1)} A(n-1,k) \qquad (1 \leq k \leq n-1)$$

the element A(n,n) being, as already mentioned, the Lucas number L_{2n-1}. Defined functions AGARTRA and AGARTRB, based on this recursion algorithm, using implicit recursion and a programmed recursion loop respectively, are for instance:

```
      ∇AGARTRA[⎕]∇
     ∇Z←AGARTRA N;K;M;R
[1]   →(N≤1)/0,Z←(N⌊1)⍴⊂⍳1
[2]   Z←AGARTRA N-1
[3]   K←⍳M←⍴↑⍉Z
[4]   R←-2×M×(1+2×M)×(↑⍉Z)÷(1+M-K)×M+K
[5]   Z←Z,⊂R,⌊0.5+(0.5×1+5*0.5)*1+2×M
     ∇
      ∇AGARTRB[⎕]∇
     ∇Z←AGARTRB N;K;M;R
[1]   →(N≤1)/0,Z←(N⌊1)⍴⊂⍳1
[2]   END:K←⍳M←⍴↑⍉Z
[3]   R←-2×M×(1+2×M)×(↑⍉Z)÷(1+M-K)×M+K
[4]   Z←Z,⊂R,⌊0.5+(0.5×1+5*0.5)*1+2×M
[5]   →(N>⍴Z)/END
     ∇
```

Benchmarks on cpu time have been done for N = 100(100)400. Results are shown below (in ms):

N	100	200	300	400
AGARTRA N	134	380	747	1230
AGARTRB N	109	331	671	1129

As N increases, AGARTRA and AGARTRB are from 4 to 17 times more efficient in cpu time than AGARTR1 and AGARTR2. The cpu times for N = 200 are of the same order as those for the other triangles treated in this paper.AGARTRB is 23% and 15% more efficient in cpu time than AGARTRA for N = 100 and N = 200 respectively, rather considerable for that domain of N.

<u>Note</u>: Agarwal gives the numbers A(n,k) the sign of $(-1)^{n-k}$, such that the sum of the numbers A(n,k) for k = 1(1)n should be 1. The numbers may be made positive, either by taking the absolute value of the explicit result of AGARTRA or AGARTRB, or by removing the minus sign from the evaluation of the local variable R in their definition.

It should be noticed that this has for effect, that it is the alter-

nating sum of the elements of the reverse of the rows of this triangle that is 1:

```
      CENTER ⊃⍕¨|¨AGARTRB 8
                  1
                 3 4
               10 20 11
             35 84 77 29
          126 336 396 261 76
      462 1320 1815 1595 836 199
   1716 5148 7865 8294 5928 2587 521
6435 20020 33033 39585 34580 20895 7815 1364

      -/¨⌽¨|¨AGARTRB 10
1 1 1 1 1 1 1 1 1 1
```

At the end of his paper (18), Agarwal poses the question if the numbers A(n,k) have a "nice" combinatorial meaning. As far as we know, such a combinatorial meaning has not yet been found.

8. <u>The DFF Triangle</u>

In 1991, G. Ferri, M. Faccio, and A.D. D'Amico published a paper introducing a triangle whose elements of the nth row are the sequence D(n,k) with k = 1...n, sequence defined by the recursion algorithm (20):

$$D(n,1) = 1$$
$$D(n,n) = 1$$
$$D(n,k) = D(n-1,k) + \sum_{r=k-1}^{n-1} D(r,k-1) \qquad (2 \leq k \leq n-1)$$

The authors call this concept a DFF triangle, the acronym DFF being derived from the initials of their names. A defined function DFFSTR1 based on the recursion algorithm shown, is for instance:

```
      ∇DFFSTR1[⎕]∇
     ∇Z←DFFSTR1 N
[1]   →(N≤1)/0,Z←(N⌊1)⍴⊂⍳1
[2]   Z←DFFSTR1 N-1
[3]   Z←Z,⊂1,(+⌿⊃Z)+(1↓↑⌽Z),0
     ∇
```

```
      CENTER ⊃⍕¨DFFSTR1 8
           1
          1 1
         1 3 1
        1 6 5 1
      1 10 15 7 1
     1 15 35 28 9 1
   1 21 70 84 45 11 1
1 28 126 210 165 66 13 1
```

The nth row contains n elements. Rows are not symmetric. The sum of the elements of the nth row is the Fibonacci number F_{2n-1}. For $n > 1$, the sum of the elements on the acending diagonals of the triangle is 2^{n-2}. The maximum of the domain of the right argument is N = 742, as D(742,332) = 1.3137629E308 and D(743,332) exceeds the largest number represeentable 1.797693135E308.

```
      +/¨DFFSTR1 10
1 2 5 13 34 89 233 610 1597 4181
      P←1 1 1 1
      (19⍴1 0)/P HORS 19
1 2 5 13 34 89 233 610 1597 4181

      10↑+/(1-⍳10)⌽(⊃DFFSTR1 10),[1]9 10⍴0
1 1 2 4 8 16 32 64 128 256
      1,2*¯1+⍳9
1 1 2 4 8 16 32 64 128 256
```

DFFSTR1 uses implicit recursion. A defined function DFFSTR2, based on the same algorithm, but using a programmed recursion loop, may be

```
      ∇DFFSTR2[⎕]∇
     ∇Z←DFFSTR2 N
[1]   →(N≤1)/0,Z←(N⌊1)⍴⊂⍳1
[2]   END:Z←Z,⊂1,(+⌿⊃Z)+(1↓↑⌽Z),0
[3]   →(N>⍴Z)/END
     ∇
```

Benchmarks on cpu time have been done for N = 100(100)300. Results are shown on next page (in ms).

The defined functions are very slow: for N = 300 the cpu time is ab-

N	100	200	300
DFFSTR1 N	525	3335	10771
DFFSTR2 N	502	3268	10741

out 11 seconds. The power of the disclose function has been exploited to calculate the sums of D(r,k-1). It is very simple, but the sums are evaluated each cycle of the loop. Less elegant, but much more efficient in cpu time, is to store those sums each cycle such that in the next cycle only the last term of the sums must be added. Defined functions DFFSTRA and DFFSTRB, based on this procedure, using implicit recursion and a programmed recursion loop respectively, are for instance:

```
      ∇DFFSTRA[⎕]∇
     ∇Z←DFFSTRA N
[1]   →(N≤1)/0,Z←(N⌊1)⍴⊂1,S←⍳0
[2]   Z←DFFSTRA N-1
[3]   S←(↑⌽Z)+S,0
[4]   Z←Z,⊂1,((1↓↑⌽Z),0)+S
     ∇
      ∇DFFSTRB[⎕]∇
     ∇Z←DFFSTRB N;S
[1]   →(N≤1)/0,Z←(N⌊1)⍴⊂1,S←⍳0
[2]   END:S←(↑⌽Z)+S,0
[3]   Z←Z,⊂1,((1↓↑⌽Z),0)+S
[4]   →(N>⍴Z)/END
     ∇
```

Benchmarks on cpu time have been done for N = 100(100)500. Results are shown below (in ms):

N	100	200	300	400	500
DFFSTRA N	93	287	588	991	1626
DFFSTRB N	70	241	521	904	1483

As N increases, DFFSTRB is from 7 to 21 times more efficient in cpu time than DFFSTR2. The cpu times for N = 200 are of the same order

as those for the other triangles treated in this paper. DFFSTRB is 33% and 19% more efficient in cpu time than DFFSTRA for N = 100 and N = 200 respectively, rather considerable for that domain of N.

Note: The coefficient of the polynomial which characterizes the transfer function of a ladder network formed by a cascade of identical uncoupled elementary cells belong to the rows of the Pascal triangle. In the case of direct coupling among interacting elementary cells forming a ladder network, the polynomial coefficients are not those belonging to the Pascal triangle but those belonging to the DFF triangle (19).

References

(1) J. De Kerf; The Pascal Triangle and APL; Les Nouvelles d'APL, No. 27, Juin 1998, pp. 77-83.

(2) J. De Kerf; De Moivre Triangles and APL; Les Nouvelles d'APL, No. 27, Juin 1998, pp. 69-76.

(3) J. De Kerf; Hoggatt Triangles and APL; Les Nouvelles d'APL, No. 28, Novembre 1998, pp. 41-50.

(4) J. De Kerf; Horadam Triangles and APL; Les Nouvelles d'APL, No. 29, Mars 1999, pp. 45-54.

(5) B.A. Bondarenko; Generalized Pascal Triangles and Pyramids - Their Fractals, Graphs, and Applications; Translated from the Russian by R.C. Bollinger; The Fibonacci Association, Santa Clara University, California, 1993.

(6) J.H. Conway and R.K. Guy; The Book of Numbers; Springer-Verlag (Copernicus), New York, New York, 1995.

(7) N.D. Thomson and R.P. Polivka; APL2 in Depth; Springer-Verlag, New York, New York, 1995.

(8) J. Shallit; A Triangle for the Bell Numbers; A Collection of Manuscripts Related to the Fibonacci Sequence; The Fibonacci Association, Santa Clara University, California, 1980; 69-71.

(9) J. De Kerf; Bell Numbers and APL; Submitted for publication in Vector (the British APL Association).

(10) E.T. Bell; Exponential Numbers; American Mathematical Monthly, Vol. 41, 1934, pp. 411-419. See also: E.T. Bell; The Iterated Exponential Integers; Annals of Mathematics, Vol. 39, 1938, pp. 539-557.

(11) D.E. Knuth; The Art of Computer Programming; Vol. 1: Fundamental Algorithms; Vol. 2: Seminumerical Algorithms; Vol. 3: Sorting and Searching; Addison-Wesley Publ. Co., Reading, Massachusetts, 1968/1969/1972.

(12) M. Abramowitz and I.A. Stegun (eds); Handbook of Mathematical Functions - With Formulas, Graphs, and Mathematical Tables; National Bureau of Standards - USA; Dover Publications, New York, New York, 1965+.

(13) C.K. Wong and T.W. Maddockx; A Generalized Pascal's Triangle; The Fibonacci Quarterly, Vol. 13, No. 2, 1975, pp. 134-136.

(14) J. De Kerf; From Fibonacci to Horadam; Vector, Vol. 15, No. 3, January 1999, pp. 122-130.

(15) L.W. Shapiro; A Catalan Triangle; Discrete Mathematics; Vol. 14, No. 1, 1976, pp. 83-90.

(16) A.K. Agarwal; On a New Kind of Numbers; The Fibonacci Quarterly, Vol. 28, No. 3, August 1990, pp. 194-199.

(17) V.E. Hoggatt; Fibonacci and Lucas Numbers; Houghton Mifflin Mathematics Enrichment Series; Houghton Mifflin Co, Boston, Massachusetts, 1969.

(18) J. De Kerf; More on "Fast Fibbing"; Vector, Vol. 3, No. 4, April 1989, pg. 120. See also: G.H. Foster; Motivating Arrays in Teaching APL; Proceedings of 5th International APL Users' Conference, Toronto, May 1973; pp. 3-1/3-8.

(19) G. Ferri; Progettazione di un microscopio ad effetto tunnel; Laurea Thesis; Università di L'Aquila, Italy, 1988.

(20) G. Ferri, M. Faccio, and A. D'Amico; A New Numerical Triangle Showing Links With Fibonacci Numbers; The Fibonacci Quarterly, Vol. 29, No. 4, November 1991, pp. 316-320. See also: A. D'Amico, M. Faccio, and G. Ferri; Determinazione della funzione di trasferimento e dell'impendenza equivalente Thevenin per reti a scala passive formate da celle elementari uguali tra di loro; Internal Research Report - Dip. Ing. Elettrica; Università di L'Aquila, Italy, 1989.

(21) ISO Document CD13751; Programming Lenguage APL - Extended; Committee Draft prepared by the APL Working Group ISO-IEC/JTC1/SC-22/WG3 - Version 1; International Organization for Standardization ISO, Geneva, Switzerland, August 1993.

(22) Dialog APL/W Reference Manual - Version 7; Dyadic Systems Ltd, Basingstoke, Hampshire, United Kingdom, 1994.

Joseph De Kerf
Rooienberg 72
B-2570 Duffel
Belgium
Tel: (32) 15 31 47 24



Le Guide des Produits et Services APL

par AFAPL

Le Guide des Produits et Services APL a pour but de fournir aux lecteurs des informations utiles sur les interprètes APL, les produits APL et les services APL disponibles sur le marché français tout en leur permettant de mieux connaître les fournisseurs.

Ces derniers sont invités à nous faire parvenir la liste de leurs produits et services de façon à figurer dans notre Guide des Produits et Services APL. La date figurant après le nom de chaque fournisseur représente la date de dernière mise à jour de leur liste de produits ou services APL.

Les prix sont indiqués en Francs Hors Taxes (port en sus).

La publication de la liste des produits et services d'un fournisseur est un service gratuit offert par AFAPL aux sociétés Membres d'AFAPL.

Abréviations utilisées dans les pages qui suivent:

N	Nouveau produit ou nouvelle version d'un produit
P	Promotion existant actuellement sur un produit

Pour nous permettre de publier la liste de vos produits et services APL, prière de faire parvenir cette dernière, imprimée, au plus tard un mois avant la parution d'un numéro au Secrétariat d'AFAPL.

Avis : Les produits et services qui n'auront pas fait l'objet d'une mise à jour (version, prix) ne pourront plus être mentionnés.

PRODUITS APL

DynaSys

Les produits ADAPTA DLS (Dynamic Logistics Systems):

ADAPTA MPS	10 KF HT	Progiciel de Calcul de Programme Directeur de Production à capacités finies.
ADAPTA PMP	85 KF HT	Module optionnel d'ADAPTA DLS MPS, pour le calcul du Plan Directeur d'Approvisionnement
ADAPTA MSC	85 KF HT	Module optionnel d'ADAPTA DLS MPS, pour le calcul des flux logistiques entre les sites de production et de distribution
ADAPTA FBS	150 KF HT	Progiciel de prévisions de ventes

LEGRAND Consultants (01/01/96) 24-98, rue Rouge de Fillol, 92200 Puteaux
Tél. : 01 46 92 09 57, Fax : 01 46 92 06 86

Dyalog APL

Dyalog APL version 6.1 pour UNIX	de 18.000 à 120.000 selon machine	Version disponible sur la plupart des plates-formes UNIX ou AIX: IBM, SUN, DEC, HP, etc...
Run-Time System pour UNIX	30%	du prix de la licence
Interface ORACLE-APL	Prix selon machine	Disponible sous UNIX pour les plates-formes IBM, SUN, HP, DEC.
Dyalog APL version 7.2 pour Windows 3.1 et 3.11	10.800 F	Possibilité d'achat groupé avec une licence Windows 95 Run-Time system gratuit
Dyalog APL version 8 pour Windows 95	10.800 F	Sortie prévue : Mars 1996 Run-Time system gratuit
Migrations vers la version 7.2		
Depuis la version 6.3	5.000 F	Possibilité d'achat groupé avec une licence Windows 95
Depuis la version 7.1	3.500 F	
Migrations vers la version Windows 95		
Depuis la version 6.3	6.500 F	
Depuis la version 7.1	5.000 F	
Prague		
Mono-utilisateur	900 F	Utilitaires d'impression et de mise à jour de tables pour Dyalog APL
Multi-utilisateurs	1.900 F	
Idem, code source fourni	2.900 F	

SYLICOM (S. Baron, G. Hervy)

P	MICROLIS version 7.0 sous Windows 3.1 et Windows 95	de 38700 F à 12400 F	Système d'Aide à la Décision avec Base de Données, Simulations et Interrogations avec calculs sur les variables.
P	Langage et Manuels J	1000F	J.3.03 (pour le reste, voir les tarifs dans le N°19)

Lescasse Consulting (12/3/99) [http://www.lescasse.com]

APL+Win version 3.0

N	APL+Win Training	**3.000 F**	**Contient 14 cours de formation avancée à APL+Win (1070 pages de documentation et 27 espaces de travail avec objets et fonctions réutilisables, dont une Base de Données Relationnelle APL)**
N	APL+Win 3.0	15.000 F	APL+ pour Windows 3.1, 95,98,NT avec support DLL, VBX, OCX, COM, DDE et licence Run Time illimitée. (CDROM)
N	Maj APL+Win 1.8-2.0 à 3.0	3.750 F	(livré sur CDROM)
N	Maj APL+Win 1.0-1.3 à 3.0	10.125 F	(livré sur CDROM)
N	Migration APL+Dos ou APL+PC (tte version) à APL+Win 3.0	11.625 F	(livré sur CDROM)

N	Migration compétitive à APL+Win 3.0	11.625 F	(livré sur CDROM)
N	APL+Link Pro 2.0	6.000 F	Accès direct depuis APL+Win aux bases de données SQL standard disposant d'un driver ODBC (Access, Paradox, Oracle, Sybase, ...) ainsi qu'à EXCEL, fichiers texte.

Voir Outils APL ci-après

APL+Dos version 6.0

N	APL+Dos 6.0	13.875 F	APL+ 2ème génération pour DOS avec Contrôles de Structures et compatibilité APL2
N	Maj APL+Dos 5.x à 6.0	2.250 F	
N	Maj APL+Dos 1.x-4.x à 6.0	6.000 F	
N	Maj APL+PC à APL+Dos 6.0	8.250 F	
N	APL+Dos 6.0 Licence Run Time illimitée	4.125 F	
N	Maj APL+Dos 6.0 Licence Run Time illimitée	2.125 F	

APL+PC version 11.0

N	APL+PC 11.0	2.650 F	APL+ 1ère génération pour DOS sans Contrôles de Structures, ni compatibilité APL2 (limité à 640K)

N	Maj APL+PC x.x à 11.0	1.125 F	

APL*PLUS II pour UNIX

	APL+Unix 5.2	de 25.600 F à 288.000F selon matériel	Interprète APL+Unix fonctionnant sous Xwindows (avec Contrôles de structure et compatibilité APL2) sur: - stations SPARC de SUN - stations DEC (excepté Alpha) - stations HP IX - IBM RS/6000
	APL+Unix 5.2 Run Time Systems	15% du prix de la licence	Version exécutable Seulement de l'interprète APL+Unix

Dyalog APL pour Windows

N	Dyalog APL 7.4+8.2	13.950 F	APL 16 et 32 bits pour Windows 3.x,95,98,NT
N	Dyalog APL 8.2	11.450 F	APL 32 bits pour Windows 95,98,NT
N	Dyalog APL 7.4	11.450 F	APL 16 bits pour Windows 3.x
N	Maj Dyalog APL		Consulter site Web http://www.lescasse.com
N	Dyalog APL Abonnement DSS		Abonnement permettant l'obtention de mises à jour de Dyalog APL par Internet pour un an Consulter site Web http://www.lescasse.com

Outils APL

N	**APL+Win Training**	3.000 F	Contient 14 cours de formation avancée à APL+Win (1070 pages de documentation et 27 espaces de travail avec objets et fonctions réutilisables, dont une Base de Données Relationnelle APL)
	DLL Parser pour APL+Win 3.0	1.450 F	Constitution automatique de déclarations de fonctions de DLL pour APLW.INI
	Delphi Form Translator pour APL+Win 3.0	1.450 F	Permet de traduire des fenêtres Delphi en fenêtres APL+Win 3.0
	Livre "Programming with APL and Delphi"	500 F	Livre de 200 pages expliquant comment exploiter Delphi dans ses développements APL
	Rain Pro 4.2 pour APL+Win	2.450 F	Progiciel graphique pour APL+Win
	New Leaf 1.5 pour APL+Win	3.950 F	Progiciel de PAO pour APL+Win
	Rain Pro 4.3 pour Dyalog APL	2.450 F	Progiciel graphique pour Dyalog APL
	New Leaf 1.5 pour Dyalog APL	3.950 F	Progiciel de PAO pour Dyalog APL
	GraphX+ChartFX pour APL+Win	5.950 F	Progiciel graphique pour APL+Win + Contrôle ChartFX

	GraphX Lite pour APL+Win	3.450 F	Progiciel graphique pour APL+Win
	AZERTY II	600 F	Support parfait du clavier français pour APL*PLUS II (V4,V5)
N	APL+Dos Toolkit II 5.2	39.000 F	Base de Données Relationnelle APL et ensemble complet d'outils de développements en APL+Dos
N	Maj APL+Dos Toolkit II 4.x à 5.2	4.000 F	Contient un générateur de rapports avec ruptures, sous-totaux; champs calculés, mise en page automatique

SERVICES APL

LEGRAND Consultants

Formation et Perfectionnement APL, Développements, Reprises d'Applications, Diffusion de DYALOG APL.

Lescasse Consulting

Régie, Conduite de Projets APL, Formations APL (débutant et perfectionnement), Portage d'application APL, Développements d'application et de produits Windows en APL, Diffusion des gammes de produits APL+ et Dyalog APL

M. Bernard MAILHOL

Travaux de Conseil, de Formation, de Conduite de Projets dans un contexte APL ou traditionnel.

SYLICOM

Travaux de Conseil et Développements en APL (tous types, y compris SHARP APL & J).

FOURNISSEURS

DynSys (07/03/96)

Mr. Augustin Holveck
10 Avenue Mendès-France
67300 SCHILTIGHEIM
tél: 03-88-19-14-14
fax:03-88-81-09-37
Messagerie Compuserve 100617,3453

LEGRAND Consultants (01/01/96)

Mr. Bernard Legrand
74-80, Rue Roque de Fillol
92800 Puteaux
tél.:01-46 92 09 57
fax:01-46 92 06 86

LESCASSE CONSULTING

Mr. Eric Lescasse
18, Rue de la Belle Feuille
92100 - BOULOGNE
tél.:01-46-05-10-76
fax 01-46-04-60-23
E-mail: eric@lescasse.com
Web site: http://www.lescasse.com

M. Bernard MAILHOL (01/01/96)

(Ingénieur Conseil)
57 av. du Dr. Durand
94110 ARCUEIL
tél.:01-46-55-91-57
fax:01-46-54-43-72

SYLICOM

Mr. Sylvain Baron
184 rue Marcel Hartmann
94200 IVRY / SEINE
tél.:01-45 21 98 50
fax 01-45 21 98 51

Bulletin d'Adhésion et de Contact

Aidez votre Association !

Communiquez-nous les coordonnées complètes d'autres personnes de votre connaissance, intéressées par le langage APL.

Nous nous ferons une obligation de leur faire parvenir un numéro gracieux des "Nouvelles d'APL".

Ils auront ainsi la possibilité d'évaluer l'intérêt pour eux de devenir Membre de notre Association. Ceci bénéficiera également à vous tous : en augmentant notre nombre d'adhérents, notre revue prendra encore plus d'essor.

Nous vous en remercions à l'avance.

A retourner au Secrétariat Général de l'Association AFAPL:

174 Bd de Charonne
F-75020 PARIS

Imprimé par ISI - 68-70, Rue des Pyrénées - 75020 PARIS

Association Francophone pour la promotion du langage APL
Association régie par la loi de 1901
174 Bd. de Charonne 75020-PARIS Tél. & Fax : 01 43 56 31 79

BULLETIN D'ADHESION 1999

Identité :

M., Mme, Mlle	
Nom	
Prénom	

Adresse professionnelle :

Société	
Code APE	
Votre fonction	
Département, Service	
Adresse	
Adresse (suite)	
Code postal	
Bureau distributeur	
Pays	
Téléphone	
FAX	
Courrier électronique	

Adresse privée :

Adresse	
Adresse (suite)	
Code postal	
Bureau distributeur	
Pays	
Téléphone	
FAX	
Courrier électronique	

Où envoyer la revue Les Nouvelles d'APL ? (cocher la case appropriée)

Adresse professionnelle	
Adresse privée	

Cotisation (accompagner l'adhésion du chèque bancaire correspondant) :

Personnelle	350 Francs (incluant l'envoi de la Revue)
Société	2 800 Francs (avec droit au Catalogue Produits)
Abonnement étranger	100 Francs (supplément pour frais d'expédition si hors CEE)

La vie de l'Association

Calendrier des Prochaines parutions

	Articles	Publicités	Parution
N° 30	15 Juin 1999	28 Juin 1999	Juillet 1999
N° 31	15 Octobre 1999	28 Octobre 1999	Novembre 1999

Les dates indiquées sont les **dates-limite de remise des articles et des publicités**. Envoyez-nous vos articles sous forme de fichier texte pur ou, de préférence, sous forme de document Word. (On peut maintenant accepter des articles écrits sur Macintosh (Word 4.1 ou 5.1) aussi bien que sur PC.) Dans tous les cas, nous souhaitons recevoir aussi une impression contrastée, type laser, de vos articles. (Eviter les envois de textes sans disquette, merci). Voir également les recommandations de la Note de la Rédaction.

Tarif des publicités

Les tarifs des publicités sont les suivants:

Type	Tarif HT
1 page complète	1000 F
1/2 page	600 F

Les typons doivent être remis, accompagnés du règlement, au plus tard un mois avant la parution du numéro dans lequel ils doivent être insérés.

Les publicités peuvent nous parvenir sous forme de pages A4 imprimées en noir et blanc sur papier blanc, de préférence sur imprimante laser. Elles seront reproduites telles quelles dans la revue de notre Association.

N° ISSN 1664-4699